Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tadej Števančec

Ravninska Delaunayeva triangulacija

diplomsko delo

univerzitetni študij računalništva in informatike

Mentor: prof. dr. Neža Mramor Kosta

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V diplomski nalogi predstavite Delaunayevo triangulacijo na podani množici točk v ravnini in opišite nekaj algoritmov za njeno konstrukcijo. Natančneje opišite naključni inkrementalni algoritem. Algoritem tudi implementirajte in preverite njegovo delovanje na več naborih naključno zgeneriranih točk. Znano je, da je pričakovana stopnja točke v Delaunayevi triangulaciji največ 6, pričakovana maksimalna stopnja točke v triangulaciji pa se z naraščajočim n asimptotično približuje vrednosti $\log^2 n$. Teoretična pričakovanja primerjajte s svojimi rezultati.

Zahvaljujem se prof. dr. Neži Mramor Kosta za mentorstvo in ves trud, strokovno pomoč ter napotke pri izdelavi diplomskega dela. Prav tako se zahvaljujem staršem za potrpljenje, podporo in omogočanje študija. Zahvala pa gre tudi sošolcem in prijateljem, brez katerih študij ne bi bil enak.

Seznam uporabljenih kratic

kratica	angleško	slovensko
DAG	Directed Acyclic Graph	usmerjeni aciklični graf

Seznam uporabljenih oznak

A_x, A_y	-	kartezijski koordinati točke A
$A(\mathcal{T})$	-	vektor kotov triangulacije $\mathcal{T}$
C	-	(očrtana) krožnica
$\mathcal{D}$	-	Delaunayevo drevo
$\mathcal{DG}(P)$	-	Delaunayev graf nad P
$\mathcal{DT}$, $\mathcal{DT}(P)$	-	Delaunayeva triangulacija oz. Delaunayeva triangulacija nad P
p, p_r	-	točka iz P s koordinatama (x, y) oz. (x_r, y_r)
$p_ip_jp_k$, $\triangle_{ijk}$	-	trikotnik z oglišči v p_i, p_j, p_k
$\overline{p_ip_j}$	-	povezava med točkama p_i in p_j
$\mathcal{P}$	-	poligon
P	-	množica točk v ravnini
$\mathcal{T}$, $\mathcal{T}(P)$	-	triangulacija oz. triangulacija nad P
U_x, U_y	-	kartezijski koordinati središča očrtane krožnice
$\mathcal{V}(p)$	-	Voronoijeva celica
$Vor(P)$	-	Voronoijev diagram nad P

Kazalo

Povzetek

Abstract

1 Uvod **1**

2 Delaunayeva triangulacija **3**

2.1 Triangulacija 3

2.2 Ravninska triangulacija 5

2.3 Voronoijev diagram 6

2.4 Delaunayeva triangulacija 9

2.4.1 Stopnja točke v Delaunayevi triangulaciji 16

3 Izgradnja Delaunayeve triangulacije **19**

3.1 Algoritmi za izgradnjo Delaunayeve triangulacije 19

3.2 Izbira algoritma 21

3.3 Naključni inkrementalni algoritem 22

3.4 Algoritem Bowyer-Watson 28

4 Implementacija algoritma **31**

4.1 Implementacija algoritma 31

4.2 Merjenje izvajalnega časa 34

4.3 Določanje najvišje in povprečne stopnje točke 35

4.4 Generator naključnih vzorcev točk 37

5 Rezultati **39**

5.1 Testno okolje in testni vzorci 39

5.2 Izvajalni čas algoritma 39

5.3 Najvišja stopnja točke 41

5.4 Povprečna stopnja točke 42

6 Razprava in sklepne ugotovitve **45**

Literatura **48**

Slike

2.1 Tloris umetnostne galerije. 4
2.2 Poligon $\mathcal{P}$ 4
2.3 Triangulacija poligona $\mathcal{P}$. 5
2.4 Primer ravninske triangulacije. 6
2.5 Descartesova delitev prostora na vrtince. 7
2.6 Voronoijev diagram in dualni Delaunayev graf. 8
2.7 Primer rabe Voronoijevega diagrama v praksi. 9
2.8 Primer rabe Delaunayeve triangulacije v praksi. 9
2.9 Thalesov izrek. 11
2.10 Optimizacija najmanjšega notranjega kota z zamenjavo povezave. 11
2.11 Pravilo očrtane krožnice. 12
2.12 Nelegalna in Delaunayeva triangulacija. 13
2.13 Središče krožnice skozi vozlišči p_i in p_j, označeno s c, leži na Voronoijevi povezavi. 15
2.14 Minimalno vpeto drevo v Delaunayevi triangulaciji. 16
2.15 Delaunayeva triangulacija točk, razporejenih okoli centralne točke. 17
2.16 Stopnja točke blizu roba konveksne ovojnice. 18

3.1 Gradnja Delaunayeve triangulacije z algoritmom s prebirno premico. 20
3.2 Združitev dveh delnih Delaunayevih triangulacij pri algoritmu s strategijo deli in vladaj. 20

3.3 Vstavitev točke v triangulacijo in legalizacija povezav pri naključnem inkrementalnem algoritmu. 21
3.4 Veliki umetni trikotnik. 22
3.5 Točka p_r se nahaja znotraj trikotnika $p_ip_kp_k$ (a) in točka p_r se nahaja na povezavi $\overline{p_ip_j}$ (b). 24
3.6 Korak vstavljanja nove točke in legalizacije povezav pri naključnem inkrementalnem algoritmu. 25
3.7 Sprememba v drevesu $\mathcal{D}$, ko se trikotnik $p_ip_jp_k$ po vstavitvi p_r razdeli na tri trikotnike. 26
3.8 Sprememba v drevesu $\mathcal{D}$ ob zamenjavi povezave $\overline{p_ip_j}$ med trikotnikoma $p_ip_jp_k$ in $p_ip_jp_l$. 27
3.9 Veliki umetni trikotnik pri algoritmu Bowyer-Watson. 28
3.10 Korak vstavljanja nove točke in legalizacije povezav pri algoritmu Bowyer-Watson. 30

4.1 Primer grafičnega izrisa našega algoritma za množico 100 točk. 35
4.2 Primer grafičnega izrisa našega algoritma za množico 10000 točk. 36

5.1 Izvajalni čas algoritma v odvisnosti od števila točk. 40

Tabele

5.1 Izvajalni čas algoritma pri podanem številu točk. 40
5.2 Najvišja stopnja točk pri podanem številu točk. 41
5.3 Maksimalna in minimalna vrednost najvišje stopnje točk pri podanem številu točk. 42
5.4 Povprečna stopnja točk pri podanem številu točk. 43

Povzetek

Naslov: Ravninska Delaunayeva triangulacija

Avtor: Tadej Ševančec

Delaunayeva triangulacija predstavlja eno izmed fundamentalnih podatkovnih struktur v računski geometriji. V diplomskem delu predstavimo ravninsko Delaunayevo triangulacijo in opišemo njeno konstrukcijo. Za izgradnjo Delaunayeve triangulacije v ravnini obstaja več vrst algoritmov, najbolj so razširjeni naključni inkrementalni. Naredili smo implementacijo algoritma Bowyer-Watson v programskem jeziku Java in preverili njegovo delovanje na več naborih naključno zgeneriranih točk. Mnogo algoritmov za izgradnjo Delaunayeve triangulacije je na tak ali drugačen način odvisnih od števila povezav, ki jim pripada posamezna točka. Primerjali smo teoretična pričakovanja za najvišjo in povprečno stopnjo točke v triangulaciji z rezultati, ki jih je vrnil naš algoritem.

Ključne besede: računska geometrija, ravninska triangulacija, ravninska Delaunayeva triangulacija, naključni inkrementalni algoritem, algoritem Bowyer-Watson, stopnja točke, pričakovana najvišja stopnja točke.

Abstract

Title: Planar Delaunay triangulation

Author: Tadej Števančec

Delaunay triangulation is one of the fundamental data structures in computational geometry. In the thesis we present the planar Delaunay triangulation and describe its construction. Several types of algorithms for building a two-dimensional Delaunay triangulation exist, the most popular are randomized incremental algorithms. We implemented the Bowyer-Watson algorithm in the programming language Java and tested it on a number of samples of randomly generated point sets. The expected degree of a vertex in a triangulation is an important parameter in many algorithms for constructing triangulations. Theoretical expectations for average and maximal vertex degrees are compared with the obtained values.

Keywords: computational geometry, planar triangulation, planar Delaunay triangulation, randomized incremental algorithm, Bowyer-Watson algorithm, vertex degree, expected maximum vertex degree.

Poglavje 1

Uvod

Področje triangulacij predstavlja enega izmed standardnih problemov v računalniški geometriji. V praksi so za nas zanimive predvsem optimalne triangulacije. Najbolj razširjena skupina optimalnih triangulacij so Delaunayeve triangulacije, ki tvorijo optimalne trikotnike glede na minimalni kot v trikotniku. Leta 1934 jih je definiral ruski matematik Boris Nikolajevič Delaunay oz. Delone [9] in danes predstavljajo eno izmed fundamentalnih podatkovnih struktur, ki se množično uporablja v računalniški grafiki, rekonstrukciji oblik in terena, interpolaciji, robotiki, računalniškem vidu, usmerjanju v omrežjih, v matematiki in splošno v znanosti. Zaradi razširjenosti področja so do danes razvili že več različnih algoritmov za izgradnjo Delaunayeve triangulacije.

Cilj diplomskega dela je preučiti tematiko Delaunayevih triangulacij v ravnini. Opisali in predstavili bomo njihove lastnosti in si ogledali postopek izgradnje. Zaradi množice obstoječih algoritmov za izgradnjo Delaunayeve triangulacije se omejimo na izbiro specifičnega algoritma iz razreda naključnih inkrementalnih. Naredili bomo delujočo implementacijo izbranega algoritma in testirali njegov izvajalni čas. V praksi bomo preverili enega izmed ekstremov v Delaunayevi triangulaciji – najvišjo stopnjo točke in ga primerjali s teoretično mejo. Ogledali si bomo tudi pričakovano stopnjo točke.

Pri diplomskem delu bomo uporabili naslednje metode raziskovanja:

- metodo deskripcije (definicija ravninske triangulacije, Voronoijevega diagrama, Delaunayeve triangulacije, opisi algoritmov itd.),
- metodo kompilacije (navedbe, sklepi, spoznanja, predlogi in ugotovitve avtorjev iz domače in tuje literature),
- metodo komparacije (primerjave med algoritmi za izgradnjo Delaunayeve triangulacije),
- kvantitativno metodo (meritve z implementiranim algoritmom in obdelava dobljenih podatkov).

V prvem delu diplomske naloge definiramo osnovne pojme, potrebne za razumevanje Delaunayeve triangulacije in predstavimo definicijo ter lastnosti Delaunaye triangulacije. Sledi poglavje, v katerem naredimo kratek pregled algoritmov za izgradnjo Delaunayeve triangulacije. Podrobneje opišemo inkrementalne konstrukcijske algoritme in izbrani algoritem. V nadaljevanju predstavimo aplikativni del diplomske naloge. Opišemo implementacijo algoritma in predstavimo metode, preko katerih smo prišli do rezultatov. V zadnjem delu predstavimo dobljene rezultate in sklepne ugotovitve.

Poglavje 2

Delaunayeva triangulacija

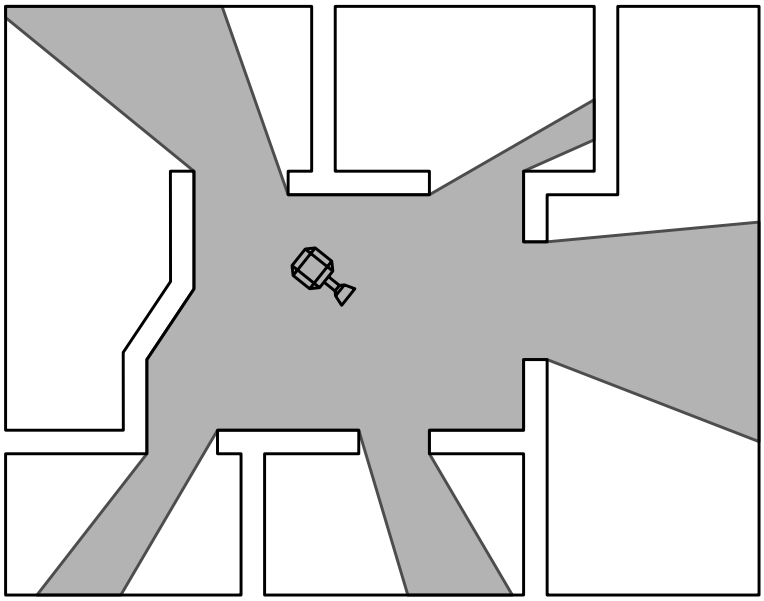
V tem poglavju si bomo najprej na praktičnem primeru ogledali pojem triangulacije in nato formalno definirali triangulacijo v ravnini. V nadaljevanju bomo predstavili Voronoijev diagram in njegovo povezavo z Delaunayevim grafom. Podrobneje bomo opisali Delaunayevo triangulacijo in njene lastnosti.

2.1 Triangulacija

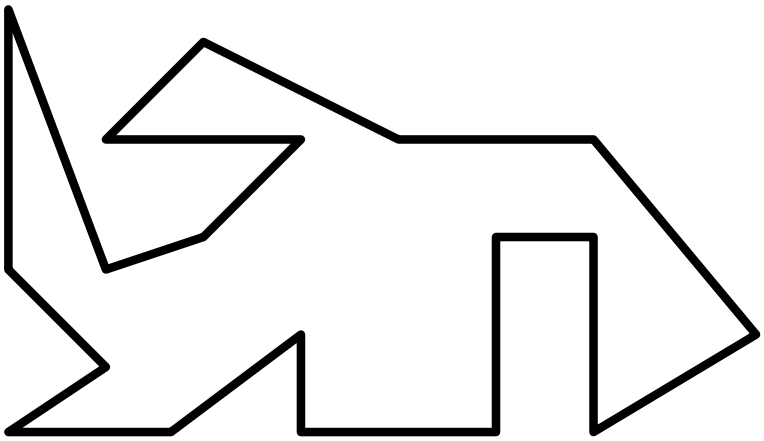
Oglejmo si pojem *triangulacije* na primeru iz prakse [8]. Vzemimo umetnostno galerijo, ki ima v prvem nadstropju en centralni prostor, v katerem se nahaja nadzorna kamera, in več sob okoli centralnega prostora. Tloris nadstropja prikazuje slika 2.1.

Vidno polje kamere pokriva omejeno področje, ki je na sliki 2.1 označeno s sivo. Zanima nas, koliko kamer bi potrebovali, da bi zavarovali celotno območje prvega nadstropja galerije.

Naj bo poligon $\mathcal{P}$ nek poligon, ki si ga lahko predstavljamo kot tloris prostora (slika 2.2). Celotno področje $\mathcal{P}$ lahko pokrijemo z vidnim poljem nadzornih kamer tako, da postavimo kamero v vsako oglišče $\mathcal{P}$. Za lažjo postavitev kamer najprej razdelimo $\mathcal{P}$ na podprostore tako, da med pari oglišč potegnemo diagonalo, kot je to prikazano na sliki 2.3.

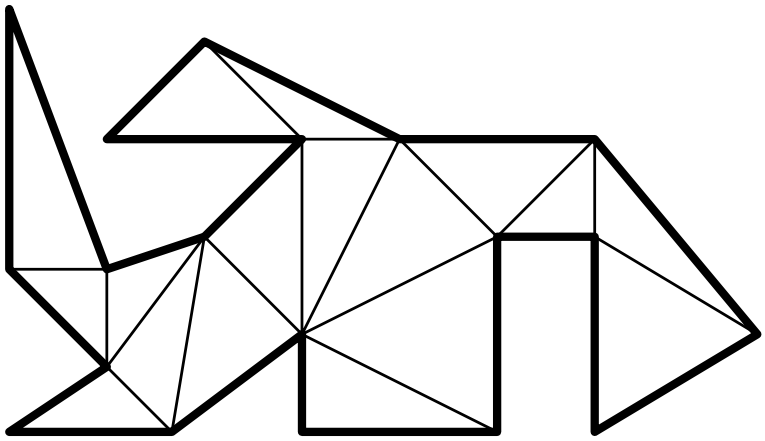


Slika 2.1: Tloris umetnostne galerije [8].



Slika 2.2: Poligon $\mathcal{P}$ [8].

Razrezu poligona na trikotnike, pri čemer je presek dveh trikotnikov prazen, ali skupna stranica, ali pa oglišče obeh, pravimo *triangulacija* poligona. Za vsak poligon obstaja triangulacija.



Slika 2.3: Triangulacija poligona $\mathcal{P}$ [8].

2.2 Ravninska triangulacija

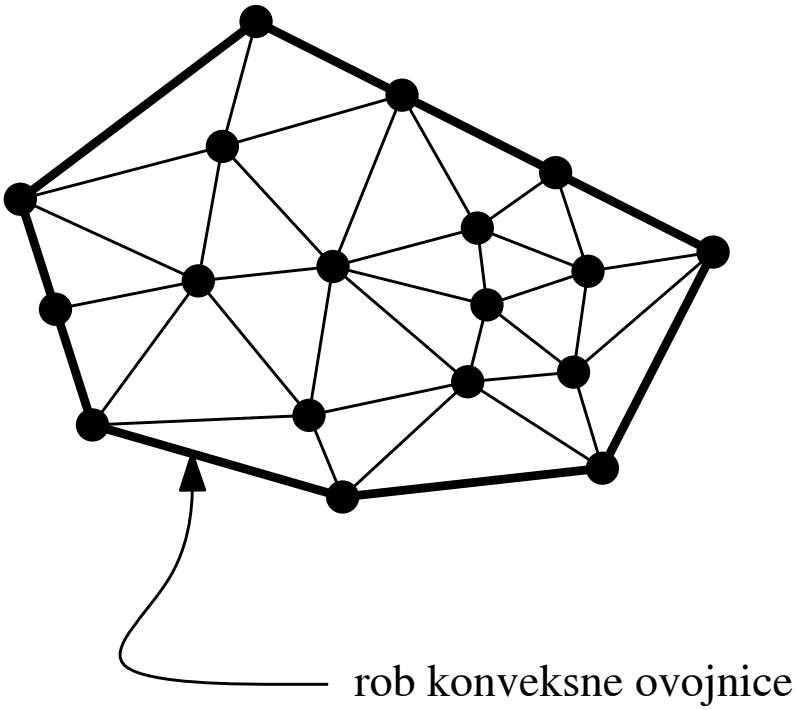
Triangulacija nad končno množico točk P v ravnini je triangulacija *konveksne ovojnice* (tudi *konveksne ogrinjače* ali *konveksne lupine*) množice P na trikotnike z oglišči v točkah množice P. Označimo jo s $\mathcal{T}(P)$.

Vsaka povezava med dvema točkama pripada natanko dvema trikotnikoma v triangulaciji, z izjemo povezav, ki ležijo na robu konveksne ovojnice. Slika 2.4 prikazuje primer ravninske triangulacije, pri čemer je rob konveksne ovojnice narisan z odebeljenimi črtami.

Vsaka triangulacija nad n točkami v ravnini ima največ $2n-5$ trikotnikov in $3n-6$ povezav [25], kar je posledica Eulerjeve formule [15], ki določa razmerje med številom oglišč n, številom povezav e in številom trikotnikov m v triangulaciji:

$$n - e + m = 1. \tag{2.1}$$

Če je k število povezav na robu konveksne ovojnice množice P, lahko izrazimo število trikotnikov in povezav triangulacije v odvisnosti od n in k. Vsak trikotnik ima tri povezave, vsaka povezava razen k robnih je v robu natanko dveh trikotnikov. Od tod sledi, da je $e = \frac{3m-k}{2} + k = \frac{3m+k}{2}$. Če



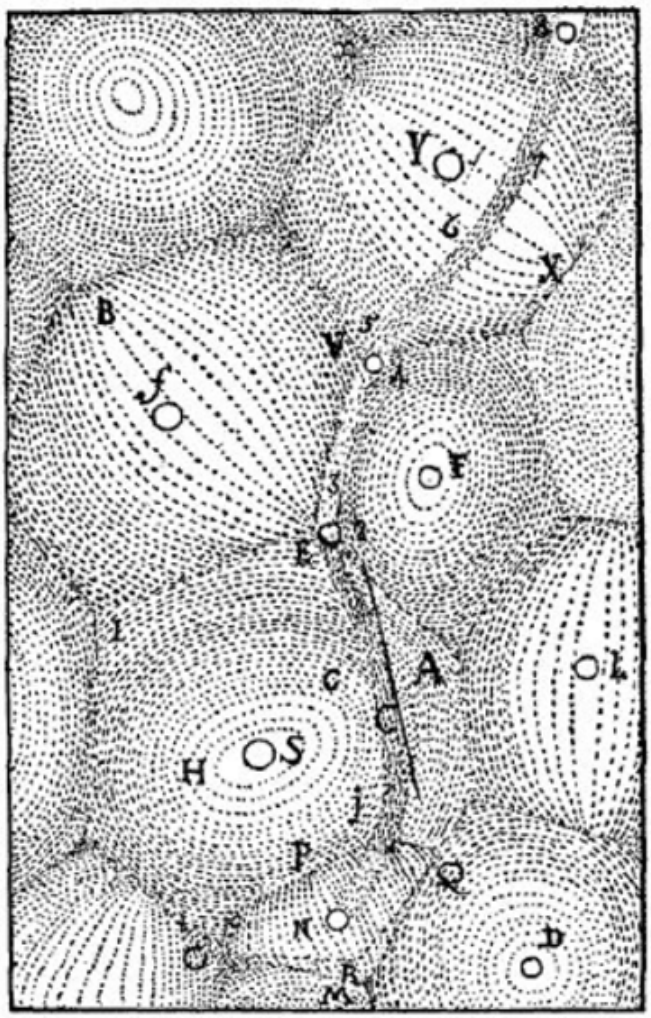
Slika 2.4: Primer ravninske triangulacije [8].

vstavimo vrednost e v Eulerjevo formulo, dobimo izračun za $m = 2n - 2 - k$ in $e = 3n - 3 - k$.

Nad množico točk v ravnini je možno zgraditi več vrst optimalnih in neoptimalnih triangulacij, za nas so kot optimalne najbolj zanimive Delaunayeve triangulacije, ki maksimizirajo najmanjši kot [13].

2.3 Voronoijev diagram

Voronoijev diagram je uporabna geometrijska struktura, ki ima svoje začetke že v 17. stoletju. Descartes je v svoji knjigi zapisal trditev [10], da je sončni sistem skupaj s svojo okolico sestavljen iz točk, ki so obdane s konveksnimi območji gravitacijske sile. Točke predstavljajo nebesna telesa, okoli katerih se znotraj pripadajočega območja giblje materija, kar je poimenoval vrtinci (slika 2.5).



Slika 2.5: Descartesova delitev prostora na vrtince [1].

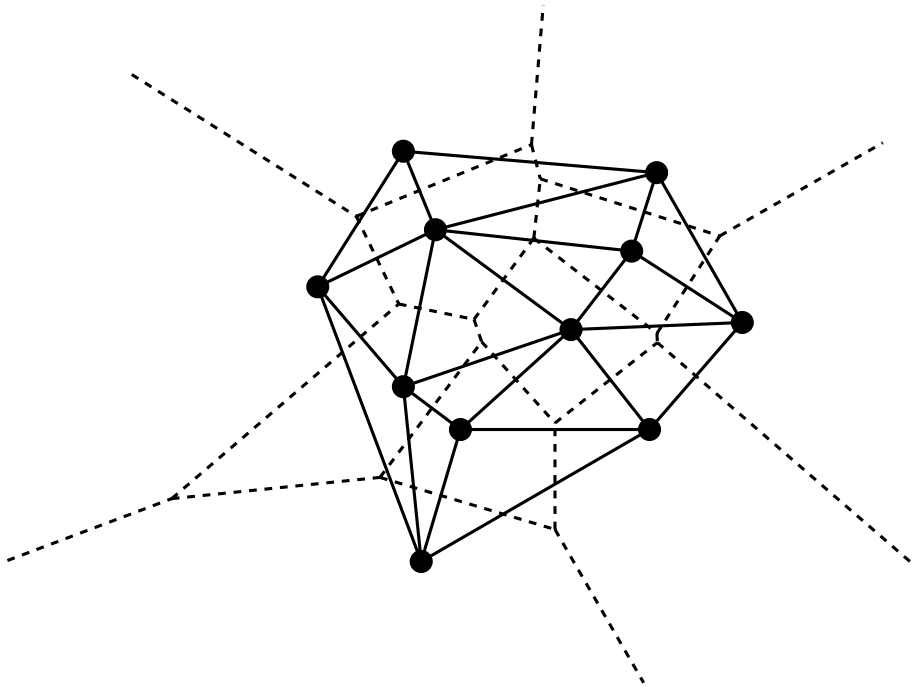
Formalno je strukturo definiral leta 1908 ukrajinski matematik Voronoi [27], po katerem je tudi dobila ime. Voronoijev diagram je tesno povezan z Delaunayevo triangulacijo, zato si oglejmo njegovo definicijo.

Naj bo P množica n točk v ravnini. Voronoijev diagram nad P definiramo [8] z razdelitvijo ravnine na n celic, tako da ima vsaka točka iz P svojo pripadajočo celico. Celicam pravimo *Voronoijeve celice*. Za Voronoijevo celico, ki pripada točki p_i iz množice P, velja:

$$\mathcal{V}(p_i) = x \in R^2 : d(x, p_i) \leq d(x, p_j), \forall i \neq j. \tag{2.2}$$

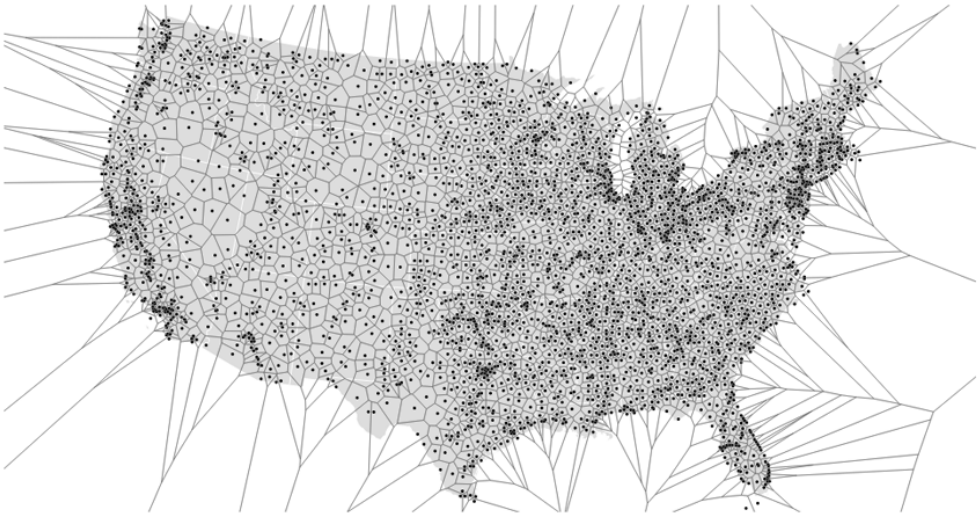
Z drugimi besedami, $\mathcal{V}(p_i)$ vsebuje vse točke v ravnini, ki so izmed vseh točk iz P najbližje p_i.

Točkam iz P pravimo *Voronoijeva vozlišča*, povezavam med njimi *Voronoijeve povezave*, stičišču povezav pa *Voronoijeve točke*. Voronoijev diagram označimo z $Vor(P)$. Če z ravnimi povezavami med seboj povežemo Voronoijeva vozlišča sosednjih Voronoijev celic, dobimo *Delaunayev graf*, ki ga označimo z $\mathcal{DG}(P)$. Pravimo, da je Voronoijev diagram *dualni graf* Delaunayevemu grafu. Slika 2.6 prikazuje povezavo med obema strukturama.



Slika 2.6: Voronoijev diagram (prekinjene črte) in dualni Delaunayev graf (krepke črte) [8].

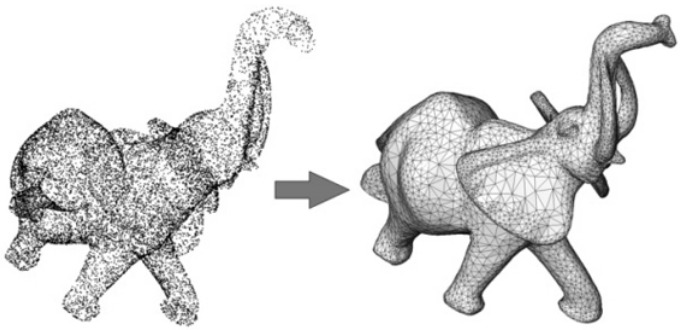
Primer praktične uporabe Voronoijevega diagrama prikazuje slika 2.7.



Slika 2.7: Voronoijev diagram prikazuje letališča v ZDA [4].

2.4 Delaunayeva triangulacija

Delaunayevo triangulacijo je leta 1934 definiral ruski matematik Delaunay [9]. Ima vrsto lepih lastnosti, zato se množično uporablja v računalniški grafiki, rekonstrukciji oblik in terena, interpolaciji, robotiki, računalniškem vidu, usmerjanju v omrežjih, v matematiki in splošno v znanosti. Primer praktične uporabe prikazuje slika 2.8.



Slika 2.8: Rekonstrukcija slona iz množice točk z Delaunayevo triangulacijo [7].

Delaunayevo triangulacijo lahko definiramo kot optimalno triangulacijo glede na najmanjši notranji kot [8]. Oglejmo si, kaj to pomeni. Vzemimo dve različni triangulaciji $\mathcal{T}$ in $\mathcal{T}'$ nad isto množico točk P, ki vsebujeta m trikotnikov in posledično $3m$ notranjih kotov. Kote v posamezni triangulaciji uredimo po velikosti v naraščajočem vrstnem redu in iz njih sestavimo vektorja kotov $A(\mathcal{T}) = (\alpha_1, \alpha_2, ..., \alpha_{3m})$ in $A(\mathcal{T}') = (\alpha'_1, \alpha'_2, ..., \alpha'_{3m})$, kjer velja $\alpha_i \leq \alpha_j$ in $\alpha'_i \leq \alpha'_j$, pri $i < j$. Če je $A(\mathcal{T})$ leksikografsko večji kot $A(\mathcal{T})'$, pravimo, da je $A(\mathcal{T})$ večji od $A(\mathcal{T})'$, kar označimo z $A(\mathcal{T}) > A(\mathcal{T}')$. Triangulacija $\mathcal{T}$ je optimalna glede na kot, če velja $A(\mathcal{T}) \geq A(\mathcal{T}')$ za vse triangulacije $\mathcal{T}'$.

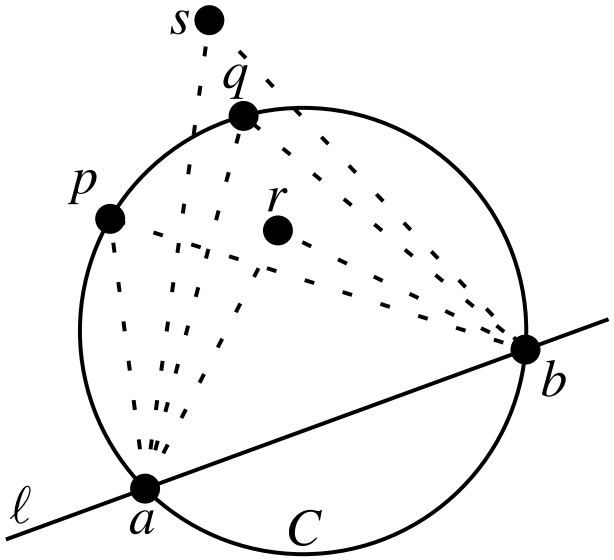
Z drugimi besedami, naj bo α_1 najmanjši kot v $\mathcal{T}$ in naj bo $A(\mathcal{T}) = (\alpha_1 \leq \alpha_2 \leq ... \leq \alpha_{3m})$ njen pripadajoči vektor kotov. Delaunayeva triangulacija je triangulacija, ki maksimizira vrednost α_1, njen vektor kotov $A(\mathcal{T})$ pa je leksikografsko največji vektor kotov. Takšne triangulacije so dobre v praktični uporabi, saj maksimizirajo ozke kote v trikotnikih, kar nam omogoča bolj naravne rezultate. Ozki trikotniki so slabi za konstrukcijo mrež in aproksimacijo v interpolaciji, rekonstrukcija terena, ki je dobljena z optimalno triangulacijo nad določeno množico točk, je bolj podobna izvirniku, kot bi bila, če bi nad množico izvedli neoptimalno triangulacijo itd.

Pri določanju optimalne triangulacije glede na kot je koristno poznati *Thalesov izrek* [8], ki pravi:

Naj bo C krožnica, l premica, ki seka C v točkah a in b, ter p, q, r in s točke, ki ležijo na isti strani l. Privzamimo, da p in q ležita na C, r leži znotraj C in s leži izven C (slika 2.9). Tedaj velja:

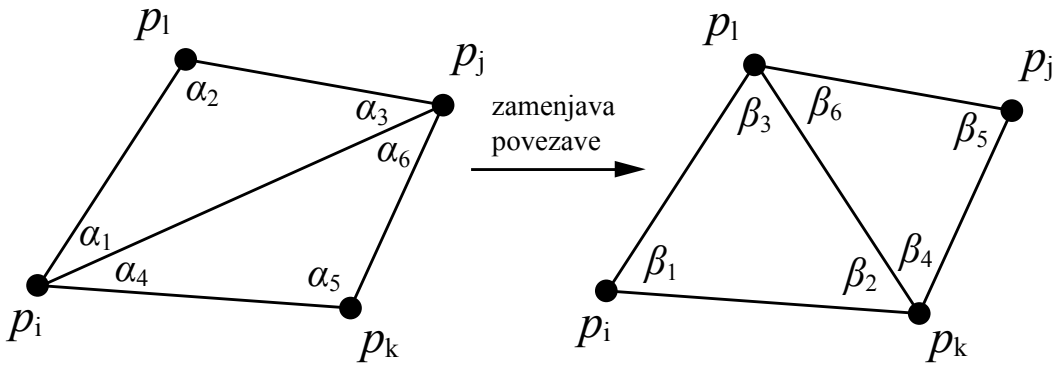
$$\measuredangle arb > \measuredangle apb = \measuredangle aqb > \measuredangle asb. \tag{2.3}$$

Vsaka povezava v triangulaciji, ki ne leži na robu konveksne ovojnice, pripada dvema trikotnikoma. Če trikotnika skupaj tvorita konveksni štirikotnik, ga lahko razpolovimo z diagonalo na dva načina: z obstoječo povezavo v triangulaciji ali s povezavo med vozliščema, ki ležita nasproti obstoječi povezavi. Zanima nas, katera povezava je "boljša". Naj bo $\overline{p_i p_j}$ povezava v



Slika 2.9: Thalesov izrek [8].

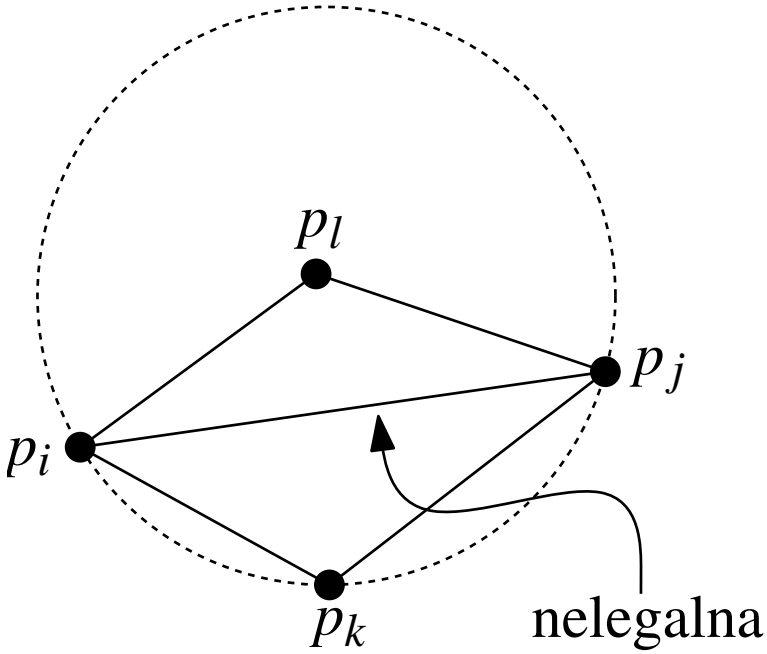
triangulaciji $\mathcal{T}$ nad množico točk P (slika 2.10), ki ne leži na robu konveksne ovojnice in je skupna trikotnikoma $p_i p_j p_k$ in $p_i p_j p_l$, ki skupaj tvorita konveksni štirikotnik. Če iz štirikotnika odstranimo povezavo $\overline{p_i p_j}$ in jo zamenjamo s povezavo $\overline{p_k p_l}$, dobimo novo triangulacijo. To operacijo imenujemo *zamenjava povezave* (angl. *edge-flip*).



Slika 2.10: Optimizacija najmanjšega notranjega kota z zamenjavo povezave.

Na sliki 2.10 smo z zamenjavo povezave $\overline{p_i p_j}$ v triangulaciji s pripadajočimi notranjimi koti $(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6)$ dobili novo triangulacijo s pripadajočimi notranjimi koti $(\beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6)$. Opazimo, da je najmanjši kot v levi triangulaciji α_1 manjši od nanjmanjšega kota v desni triangulaciji β_6. Povezavi $\overline{p_i p_j}$ pravimo *nelegalna*, če velja: $\min \alpha_i < \min \beta_i, 1 \leq i \leq 6$. Drugače povedano, povezava je nelegalna, če lahko lokalno povečamo najmanjši kot z zamenjavo te povezave.

Da nam ni potrebno računati kotov v triangulaciji, obstaja kriterij, ki je bolj prikladen za preverjanje legalnosti povezav. Pravimo mu *pravilo očrtane krožnice* (angl. *circumcircle property*) ali *Delaunayevo pravilo* [8] (slika 2.11):



Slika 2.11: Pravilo očrtane krožnice [8].

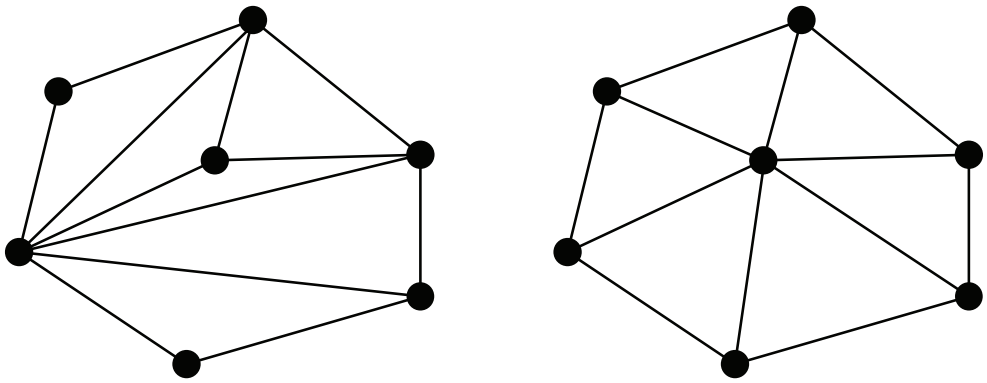
Naj bo povezava $\overline{p_i p_j}$ skupna trikotnikoma $p_i p_j p_k$ in $p_i p_j p_l$, in naj bo C krožnica skozi p_i, p_j in p_k. Povezava $\overline{p_i p_j}$ je nelegalna natanko tedaj, ko točka p_l leži znotraj C. Nadalje, če točke p_i, p_j, p_k in p_l skupaj tvorijo konveksni štirikotnik in ne ležijo na skupni krožnici, je natanko ena izmed povezav $\overline{p_i p_j}$

in $\overline{p_k p_l}$ *nelegalna.*

Opazimo, da je kriterij simetričen za točki p_k in p_l: p_l leži znotraj krožnice skozi p_i, p_j, p_k natanko tedaj, ko p_k leži znotraj krožnice skozi p_i, p_j, p_l. Če vse štiri točke ležijo na isti krožnici, sta povezavi $\overline{p_i p_j}$ in $\overline{p_k p_l}$ obe legalni. Triangulaciji, ki ne vsebuje nobene nelegalne povezave, pravimo *legalna triangulacija*.

Delaunayevo triangulacijo lahko sedaj definiramo še s pomočjo Delaunayevega pravila. Triangulacija $\mathcal{T}(P)$ nad končno množico točk P je Delaunayeva triangulacija, če za vsak trikotnik v $\mathcal{T}(P)$ velja, da se znotraj njegove očrtane krožnice ne nahaja nobena druga točka iz P. Označimo jo z $\mathcal{DT}(P)$. Če v množici točk P ne obstajajo štiri točke, ki ležijo na isti krožnici, je Delaunayeva triangulacija nad P ena sama.

Slika 2.12 prikazuje razliko med nelegalno in Delaunayevo triangulacijo.



Slika 2.12: Nelegalna (a) in Delaunayeva triangulacija (b).

Lawson je dokazal [20], da iz poljubne triangulacije z zamenjavami povezav lahko dobimo poljubno drugo triangulacijo. Od tod sledi, da lahko z zamenjavo vseh nelegalnih povezav dobimo Delaunayevo triangulacijo. Algoritem, ki za vsak par trikotnikov v triangulaciji, ki skupaj tvorita konveksen štirikotnik, izvede legalizacijo povezave, dokler triangulacija ne vsebuje več

nelegalnih povezav, je predstavljen v izpisu 1.

Izpis 1 LegalizirajTriangulacijo($\mathcal{T}$)

Vhod. Triangulacija $\mathcal{T}$ nad množico točk P.

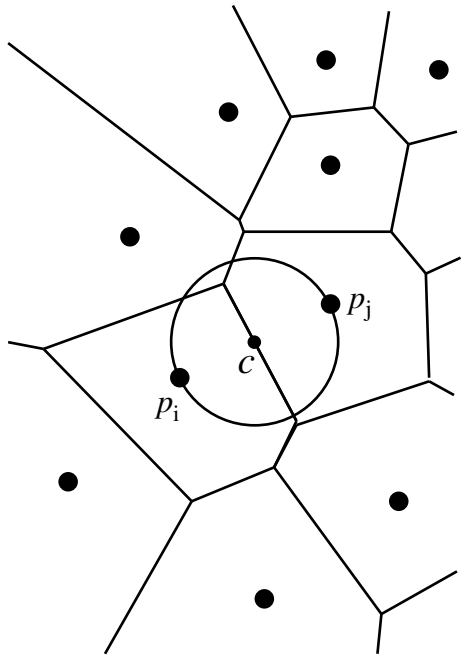
Izhod. Legalna triangulacija nad P.

1: **dokler** $\mathcal{T}$ vsebuje nelegalno povezavo $\overline{p_i p_j}$
2: **naredi** * zamenjavo povezave $\overline{p_i p_j}$ *
3: Naj bosta $p_i p_j p_k$ in $p_i p_j p_l$ trikotnika s skupno povezavo $\overline{p_i p_j}$.
4: Odstrani $\overline{p_i p_j}$ iz $\mathcal{T}$ in dodaj $\overline{p_k p_l}$.
5: **vrni** $\mathcal{T}$

Delaunayeva triangulacija ima še nekaj dobrih lastnosti, ki jih je vredno omeniti:

- *Lastnost prazne krožnice (angl. empty circle property).*
  Podobno kot velja lastnost prazne očrtane krožnice za trikotnike v Delaunayevi triangulaciji, velja tudi za povezave lastnost prazne krožnice. Naj bo $\mathcal{DT}(P)$ Delaunayeva triangulacija nad končno množico točk P. Za vsako povezavo $\overline{p_i p_j}$ v $\mathcal{DT}(P)$ obstaja krožnica skozi točki p_i in p_j, za katero velja, da nobena izmed ostalih točk iz P ne leži znotraj te krožnice. Pravilnost lastnosti zlahka opazimo, če si ogledamo Voronoijev diagram $Vor(P)$ nad množico P, v katerem narišemo krožnico skozi dve Voronoijevi vozlišči. Središče te krožnice leži na povezavi v $Vor(P)$, saj je razdalja od središča do obeh točk, ki ležita na krožnici, enaka, znotraj krožnice pa ne leži nobeno drugo vozlišče v $Vor(P)$ (slika 2.13).

- *Lastnost para najbližjih točk (angl. closest pair property).*
  Naj bosta točki p_i in p_j dve točki v končni množici P, ki sta si med vsemi točkami v P najbližje. Če nad P zgradimo Delaunayevo triangulacijo $\mathcal{DT}(P)$, bosta točki p_i in p_j v $\mathcal{DT}(P)$ *sosednji*. Pravilnost te lastnosti lahko hitro dokažemo z lastnostjo prazne krožnice. Znotraj krožnice skozi točki p_i in p_j v $\mathcal{DT}(P)$ se ne nahaja nobena druga točka iz P, zato sta p_i in p_j sosednji.

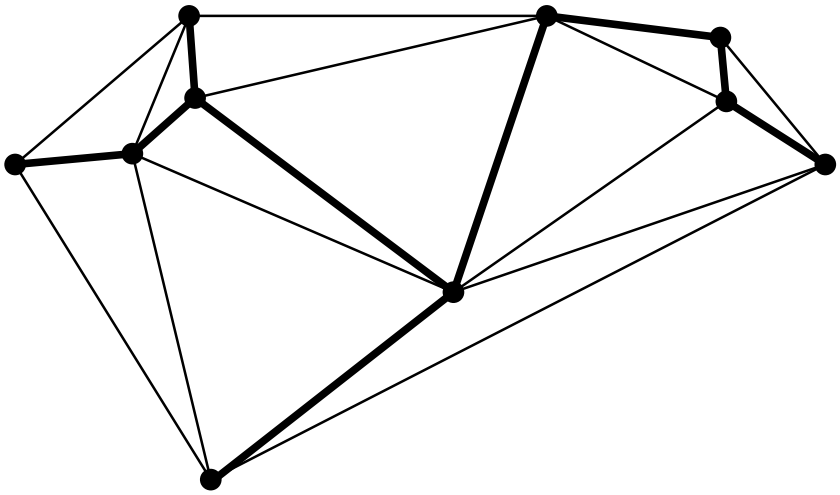


Slika 2.13: Središče krožnice skozi vozlišči p_i in p_j, označeno s c, leži na Voronoijevi povezavi.

- *Minimalno vpeto drevo (angl. minimum spanning tree) je podgraf v Delaunayevi triangulaciji.*

  Naj bo $\mathcal{DT}(P)$ Delaunayeva triangulacija nad končno množico točk P. Potem velja, da je minimalno vpeto drevo v P podgraf v $\mathcal{DT}(P)$. Vsaka povezava, ki je del minimalnega vpetega drevesa v P, je tudi povezava v $\mathcal{DT}(P)$. Obratno velja za vsako povezavo, ki ni del minimalnega vpetega drevesa v P, da tudi ni del $\mathcal{DT}(P)$. Dokaz lahko izpeljemo iz lastnosti prazne krožnice Delaunayevih triangulacij in acikličnosti minimalnih vpetih dreves. Naj bo $\overline{p_i p_j}$ povezava med dvema točkama v končni množici P, ki ni povezava v $\mathcal{DT}(P)$. Preko lastnosti prazne krožnice opazimo, da se znotraj krožnice skozi točki p_i in p_j mora nahajati še neka druga točka p_k iz P. Od tod sledi, da sta razdalji med p_k in p_i ter p_k in p_j krajši od razdalje med p_i in p_j, zato je razdalja med p_i in p_j najdaljša razdalja v cikličnem podgrafu, sestavljenem iz

točk p_i, p_j in p_k. Za minimalno vpeto drevo velja, da je acikličen podgraf nekega grafa. Ker je povezava $\overline{p_i p_j}$ del cikličnega podgrafa, iz tega sledi, da povezava $\overline{p_i p_j}$ ne more biti povezava v minimalnem vpetem drevesu. Slika 2.14 prikazuje primer minimalnega vpetega drevesa v Delaunayevi triangulaciji.



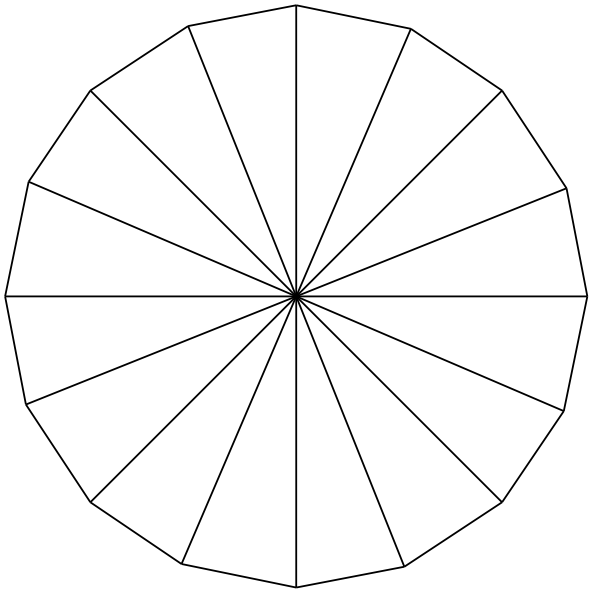
Slika 2.14: Minimalno vpeto drevo (krepke črte) v Delaunayevi triangulaciji.

2.4.1 Stopnja točke v Delaunayevi triangulaciji

Mnogo algoritmov za izgradnjo Delaunayeve triangulacije je na tak ali drugačen način odvisnih od števila povezav, ki jim pripada posamezna točka, torej od *stopenj točke* v triangulaciji.

Naj bo $\mathcal{DT}(P)$ Delaunayeva triangulacija nad množico točk P z močjo n. Najvišja možna stopnja točke v $\mathcal{DT}(P)$ je $n-1$. Zgornjo mejo dosežemo v primeru, ko je $n-1$ točk razporejenih okoli centralne točke, kot to prikazuje slika 2.15.

V splošnem je ta meja zelo pesimistična, saj jo dobimo le v najslabšem primeru, zato nas v praktičnih primerih zanima bolj realistična pričakovana meja. V literaturi najdemo malo omemb pojma najvišje stopnje točke v Delaunayevi triangulaciji. Bern, Eppstein in Yao [3] so dokazali, da pričakovana

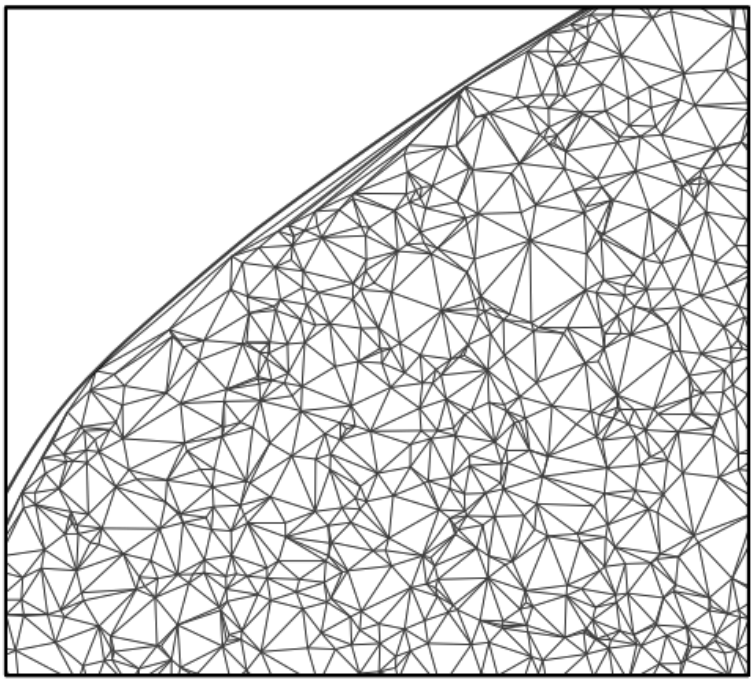


Slika 2.15: Delaunayeva triangulacija točk razporejenih okoli centralne točke [6].

najvišja stopnja točke v Delaunayevi triangulaciji v ravnini, pri čemer so točke razporejene s homogenim Poissonovim procesom na intervalu $[0, \sqrt{n}]^2$, znaša $\frac{\log n}{\log \log n}$. V praksi se izkaže, da meja ni dovolj natančna, saj je le približek, omejen na razporeditev točk s Poissonovim procesom, prav tako pa ne upošteva, da imajo v praksi točke, ki ležijo pri robu konveksne ovojnice triangulacije, mnogokrat višjo stopnjo od pričakovane, kot to prikazuje slika 2.16.

Broutin, Devillers in Hemsley so dokazali [6], da je za naključno razporeditev točk v ravnini v Delaunayevi triangulaciji bolj natančna pričakovana najvišja stopnja točke $\log^{2+\varepsilon} n$, za vsak $\varepsilon > 0$, ko gre število točk n proti neskončnosti.

Iz Eulerjeve formule (2.1) je razvidno, da ima Delaunayeva triangulacija $\mathcal{DT}$ nad množico točk moči n manj kot $3n$ povezav in $2n$ trikotnikov. Ker posamezna povezava vsebuje natanko dve točki, je vsota stopenj točke za vsa vozlišča v $\mathcal{DT}$ manjša od $6n$. Če to vsoto delimo s številom vseh točk v $\mathcal{DT}$,



Slika 2.16: Stopnja točke blizu roba konveksne ovojnice [6].

dobimo *povprečno stopnjo točke* v $\mathcal{DT}$, ki je manjša od 6.

Poglavje 3

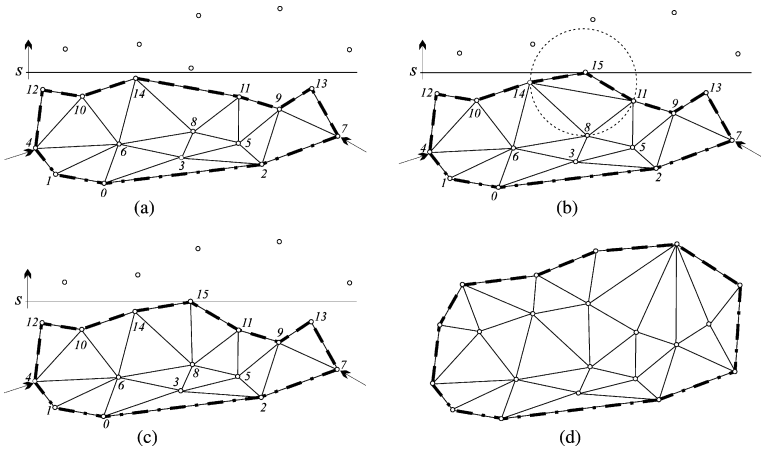
Izgradnja Delaunayeve triangulacije

V tem poglavju bomo pripravili kratek pregled najbolj razširjenih algoritmov za izgradnjo Delaunayeve triangulacije in pojasnili izbiro algoritma za našo implementacijo. Podrobneje si bomo ogledali inkrementalne konstrukcijske algoritme in opisali izbrani algoritem Bowyer-Watson.

3.1 Algoritmi za izgradnjo Delaunayeve triangulacije

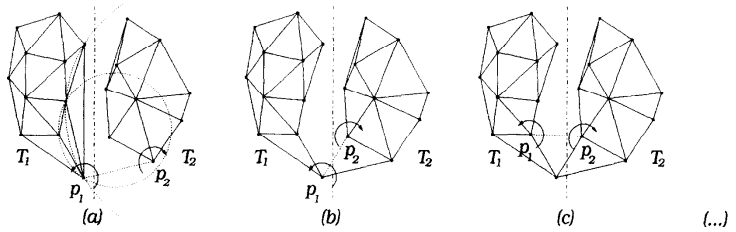
Za izgradnjo Delaunayeve triangulacije danes obstaja več vrst algoritmov. Njihova pričakovana časovna zahtevnost je $\mathcal{O}(n \log n)$ [2]. Med pomembnejše sodijo:

- *Algoritmi s prebirno premico.*
  Algoritem je prvi razvil Fortune [16] in temelji na sprehajanju prebirne premice preko ravnine s točkami. Premica postopoma jemlje točke v naraščajočem vrstnem redu po eni izmed koordinat točke in jih dodaja v Delaunayevo triangulacijo, dokler se ne sprehodi preko celotne množice. Slika 3.1 prikazuje primer delovanja algoritma.



Slika 3.1: Gradnja Delaunayeve triangulacije z algoritmom s prebirno premico [29].

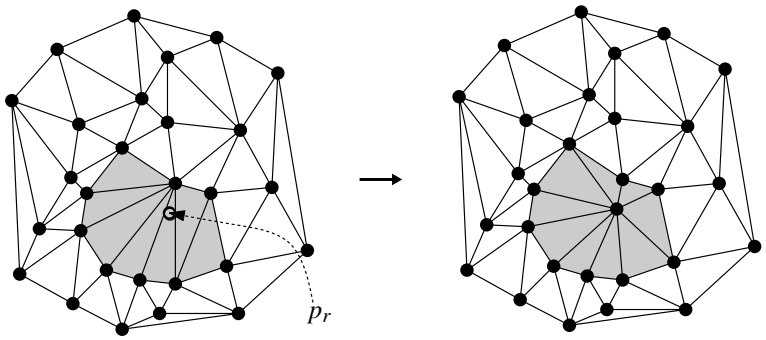
- *Algoritmi s strategijo deli in vladaj.*
  Prvotno verzijo algoritma sta razvila Lee in Schachter [23]. Algoritmi te vrste temeljijo na razdelitvi glavnega problema na podprobleme. Ravnina s točkami se tako razdeli na več ločenih delov, nad katerimi se rekurzivno zgradi Delaunayeva triangulacija. Triangulacije se nato združijo v celoto. Slika 3.2 prikazuje primer združitve dveh delnih Delaunayevih triangulacij.



Slika 3.2: Združitev dveh delnih Delaunayevih triangulacij [12].

- *Inkrementalni konstrukcijski algoritmi.*
  Algoritmi delujejo po principu zaporedne gradnje Delaunayeve trian-

gulacije za vsak posamezen korak, ki sestoji iz vstavljanja nove točke v obstoječo Delaunayevo triangulacijo in delitve trikotnika, v katerega pade točka, dokler niso obdelane vse točke. Med najbolj razširjene verzije algoritma sodijo Lawsonov [19], Bowyer-Watsonov [5, 28] in algoritem, ki so ga razvili Guibas, Knuth in Sharir [17]. Primer vstavitve točke v triangulacijo in legalizacije povezav prikazuje slika 3.3.



Slika 3.3: Vstavitev točke v triangulacijo in legalizacija povezav [8].

3.2 Izbira algoritma

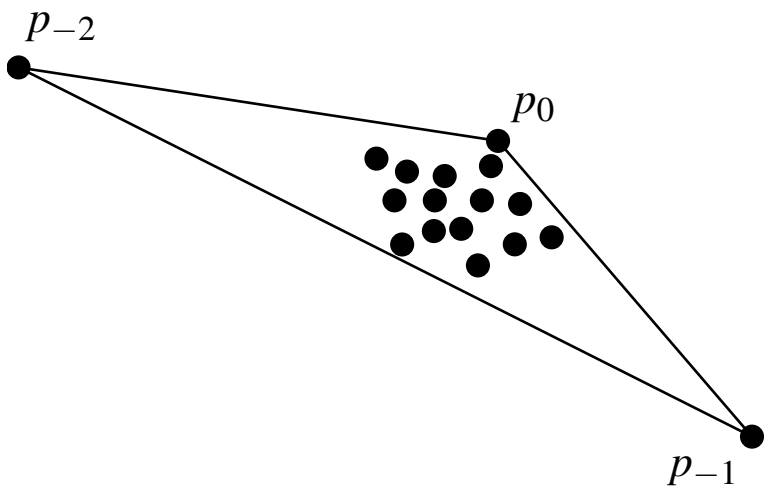
Implementacija algoritmov s prebirno premico in algoritmov s strategijo deli in vladaj zahteva kompleksno programsko kodo, zato so bolj razširjeni inkrementalni konstrukcijski algoritmi [17]. Prednost teh algoritmov je lažja implementacija ob primerni izbiri podatkovne strukture, prav tako ni potrebno, da imamo množico točk vnaprej znano. Algoritmi so naključni, kar pomeni, da se njihovo kompleksnost lahko analizira z orodji iz verjetnostne teorije [26].

Po primerjavi psevdokode algoritmov Lawson in Bowyer-Watson smo se odločili za izbiro slednjega, ker nam je deloval nekoliko lažji za implementacijo.

3.3 Naključni inkrementalni algoritem

Za lažje razumevanje samega postopka gradnje Delaunayeve triangulacije si najprej oglejmo naključni inkrementalni algoritem po vzoru Lawsona, kot je predstavljen v [8]. Algoritem kot primarni kriterij upošteva pravilo prazne očrtane krožnice, za tvorjenje Delaunayevih trikotnikov pa rekurzivno uporablja zamenjavo vseh nelegalnih povezav. Izpis 2 prikazuje psevdokodo algoritma.

Algoritem se začne s tvorbo dovolj velikega umetnega trikotnika, ki zaobjame vse točke iz množice P, nad katero gradimo Delaunayevo triangulacijo. Za oglišča umetnega trikotnika določimo dve umetni točki p_{-1} in p_{-2} ter $p_0 \in P$, za katero velja, da je najvišja točka v P (slika 3.4).



Slika 3.4: Veliki umetni trikotnik [8].

Tako bomo gradili Delaunayevo triangulacijo nad $P \cup \{p_{-1}, p_{-2}\}$, zato bo v zadnjem koraku potrebno odstraniti točki p_{-1} in p_{-2} ter vse povezave, ki jima pripadajo. Umetni točki lahko obravnavamo simbolično in v skladu s tem dodamo v algoritem pravila, ki obravnavajo umetni točki kot dejanski točki v P, lahko pa jima določimo dovolj veliki koordinati, da ne vplivata na zgradbo Delaunayeve triangulacije nad P. Inicializacija triangulacije se

Izpis 2 Algoritem: DelaunayevaTriangulacija(P) [8]

Vhod. Množica P dolžine $n+1$ točk v ravnini.

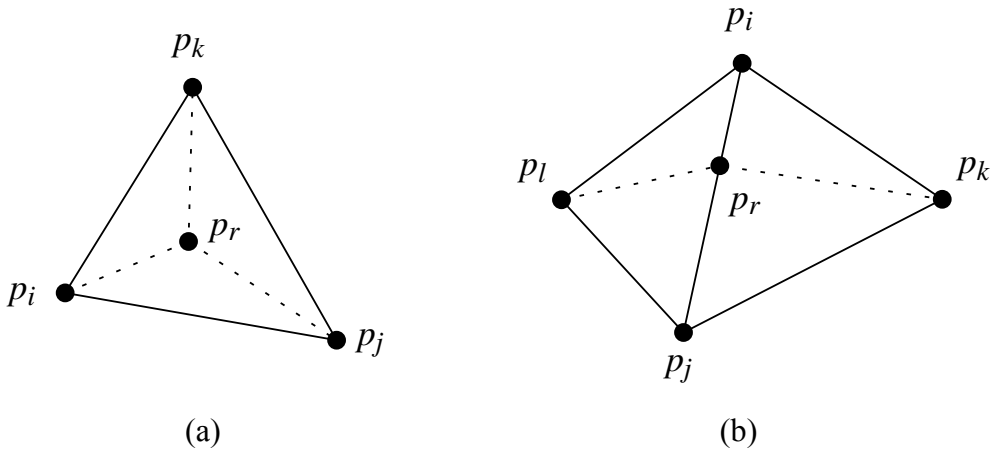
Izhod. Delaunayeva triangulacija nad P.

1: Naj bo p_0 leksikografsko najvišja točka v P, ki je najbolj desno med vsemi točkami z največjo y-koordinato.
2: Naj bosta p_{-1} in p_{-2} dve točki v $\mathbb{R}^2$ dovolj daleč, da je P zajeta v trikotniku $p_0p_{-1}p_{-2}$.
3: Inicializirajmo $\mathcal{T}$ kot triangulacijo, ki vsebuje en sam trikotnik $p_0p_{-1}p_{-2}$.
4: Naredimo naključno permutacijo $p_1, p_2, ..., p_n$ nad $P \setminus p_0$.
5: **za** $r \leftarrow 1$ **do** n
6: **naredi** * Vstavi p_r v $\mathcal{T}$: *
7: Poišči trikotnik $p_ip_jp_k \in \mathcal{T}$, ki vsebuje p_r.
8: **če** p_r leži znotraj trikotnika $p_ip_jp_k$
9: **potem** Dodaj povezave iz p_r v tri vozlišča $p_ip_jp_k$ in s tem razdeli $p_ip_jp_k$ na tri trikotnike.
10: *LegalizirajPovezavo*$(p_r, \overline{p_ip_j}, \mathcal{T})$
11: *LegalizirajPovezavo*$(p_r, \overline{p_jp_k}, \mathcal{T})$
12: *LegalizirajPovezavo*$(p_r, \overline{p_kp_i}, \mathcal{T})$
13: **drugače** * p_r leži na povezavi med oglišči $p_ip_jp_k$, recimo povezava $\overline{p_ip_j}$ *
14: Dodaj povezave iz p_r v p_k in v tretje vozlišče p_l, ki se nahaja v drugem trikotniku s skupno povezavo $\overline{p_ip_j}$. S tem razdeli trikotnika s skupno $\overline{p_ip_j}$ na štiri trikotnike.
15: *LegalizirajPovezavo*$(p_r, \overline{p_ip_l}, \mathcal{T})$
16: *LegalizirajPovezavo*$(p_r, \overline{p_lp_j}, \mathcal{T})$
17: *LegalizirajPovezavo*$(p_r, \overline{p_jp_k}, \mathcal{T})$
18: *LegalizirajPovezavo*$(p_r, \overline{p_kp_i}, \mathcal{T})$
19: Odstrani p_{-1}, p_{-2} in vse pripadajoče povezave iz $\mathcal{T}$.
20: **vrni** $\mathcal{T}$

začne s trikotnikom $p_0p_{-1}p_{-2}$, nato pa iterativno dodajamo v triangulacijo

preostale točke iz P in vzdržujemo Delaunayevo triangulacijo množice točk, ki smo jih že vstavili v algoritem. Algoritem je naključen, kar pomeni, da točke dodajamo v naključnem vrstnem redu.

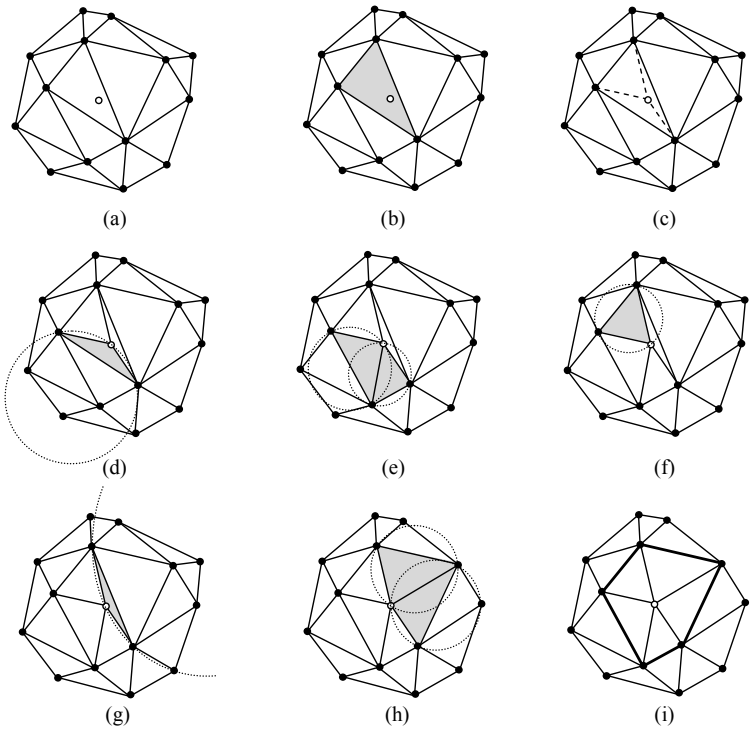
Najpomembnejša logika algoritma je zajeta v postopku vstavljanja nove točke, zato si podrobneje oglejmo ta korak (slika 3.6). Naj bo p_r točka iz P. Najprej v triangulaciji poiščemo trikotnik, znotraj katerega se nahaja p_r. Pri tem naletimo na dva možna primera: p_r se nahaja znotraj trikotnika, ali pa se p_r nahaja na povezavi, ki si jo dva trikotnika delita. Slika 3.5 prikazuje oba primera.



Slika 3.5: Točka p_r se nahaja znotraj trikotnika $p_ip_kp_k$ (a) in točka p_r se nahaja na povezavi $\overline{p_ip_j}$ (b) [8].

V prvem primeru razdelimo trikotnik na tri nove trikotnike, tako da točko p_r povežemo z vsemi tremi oglišči trikotnika. V drugem primeru razdelimo vsakega od trikotnikov, ki si delita povezavo, na kateri je točka p_r, na dva dela tako, da točko p_r povežemo z nasprotnim ogliščem.

Rezultat delitve je triangulacija, ki ni nujno več Delaunayeva. Potrebno je preveriti, ali nove povezave ustrezajo pravilu očrtane krožnice in ali ni kakšna od prej obstoječih legalnih povezav postala nelegalna, kar je potrebno popraviti. To naredimo tako, da kličemo proceduro *LegalizirajPovezavo* za vsako morebitno nelegalno povezavo. Procedura rekurzivno v triangulaciji zame-



Slika 3.6: V obstoječo $\mathcal{DT}$ vstavimo novo točko (a). Poiščemo trikotnik, znotraj katerega se nahaja nova točka (trikotnik je obarvan sivo) (b) in ga razdelimo na nove trikotnike (c). Preverimo, če sosednji trikotniki ustrezajo lastnosti prazne krožnice in zamenjamo povezave, če je potrebno (d) do (h). Nova $\mathcal{DT}$, ki vsebuje vstavljeno točko (i) [22].

nja vse nelegalne povezave z legalnimi. Psevdokoda procedure je prikazana v izpisu 3.

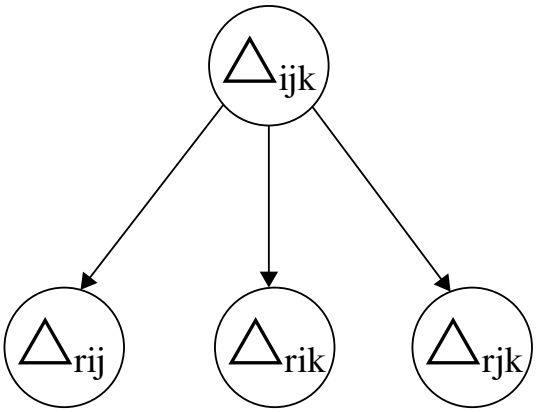
Algoritem za iskanje trikotnika, v katerem se nahaja p_r, sočasno ob gradnji Delaunayeve triangulacije gradi podatkovno strukturo $\mathcal{D}$, ki je *usmerjen aciklični graf* (angl. Directed Acyclic Graph ali DAG). Strukturi pravimo tudi *Delaunayevo drevo*. Za vsak trikotnik v triangulaciji ustvarimo vozlišče v drevesu $\mathcal{D}$. Listi v $\mathcal{D}$ predstavljajo trikotnike trenutne triangulacije, notranja vozlišča pa trikotnike, ki so bili del triangulacije v eni izmed prejšnjih iteracij algoritma. Med vozlišči in listi urejamo kazalce, ki kažejo potek de-

Izpis 3 Procedura: LegalizirajPovezavo $p_r, \overline{p_i p_j}, \mathcal{T}$ [8]

```
1: * p_r je vstavljena točka in p_i p_j je povezava v T, ki jo je morda potrebno
   zamenjati. *
2: če p_i p_j je nelegalna
3:     potem Naj bo trikotnik p_i p_j p_k sosednji trikotniku p_r p_i p_j, trikotnika
4:         imata skupno povezavo p_i p_j.
5:         * Zamenjaj p_i p_j: * Nadomesti p_i p_j s p_r p_k.
6:         LegalizirajPovezavo(p_r, p_i p_k, T)
7:         LegalizirajPovezavo(p_r, p_k p_j, T)
```

litve trikotnikov. Drevo $\mathcal{D}$ inicializiramo kot DAG z enim samim vozliščem, ki ustreza umetnemu trikotniku $p_0 p_{-1} p_{-2}$.

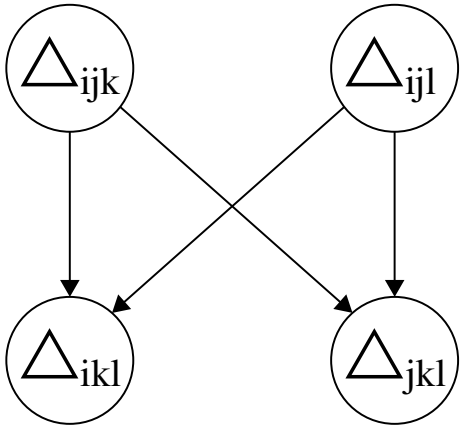
Primer spreminjanja drevesa $\mathcal{D}$, ko v določeni iteraciji točka p_r razdeli trikotnik $p_i p_j p_k$ na tri nove trikotnike, prikazuje slika 3.7. Trikotnik $p_i p_j p_k$, ki je bil prej list drevesa $\mathcal{D}$, postane vozlišče s kazalci na tri naslednike, ki so novi listi v $\mathcal{D}$.



Slika 3.7: Sprememba v drevesu $\mathcal{D}$, ko se trikotnik $p_i p_j p_k$ po vstavitvi p_r razdeli na tri trikotnike.

Podobna sprememba sledi ob zamenjavi povezave. Recimo, da sta $p_i p_j p_k$ in $p_i p_j p_l$ trikotnika z nelegalno skupno povezavo $\overline{p_i p_j}$. Ob zamenjavi povezave ustvarimo v drevesu $\mathcal{D}$ lista za nova trikotnika, vozlišča pripadajoča

trikotnikoma $p_ip_jp_k$ in $p_ip_jp_l$ pa dobita kazalec na nova lista, kar prikazuje slika 3.8.



Slika 3.8: Sprememba v drevesu $\mathcal{D}$ ob zamenjavi povezave $\overline{p_ip_j}$ med trikotnikoma $p_ip_jp_k$ in $p_ip_jp_l$.

Vozlišče lahko dobi največ tri naslednike, za iskanje trikotnika, v katerem je točka p_r, pa se je potrebno sprehoditi samo skozi trikotnike, ki vsebujejo točko p_r, zato lahko to opravimo v linearnem času. To storimo tako, da začnemo z iskanjem pri korenu drevesa $\mathcal{D}$, nato pa nadaljujemo v tistem nasledniku, znotraj katerega se nahaja p_r in vse tako naprej, dokler ne dosežemo lista v $\mathcal{D}$, ki predstavlja iskani trikotnik.

Opisani algoritem za gradnjo Delaunayeve triangulacije nad množico točk dolžine n porabi v praksi največ $\mathcal{O}(n \log n)$ časa, s porabo $\mathcal{O}(n)$ časa, namenjenemu za uporavljanje s podatkovno strukturo [17].

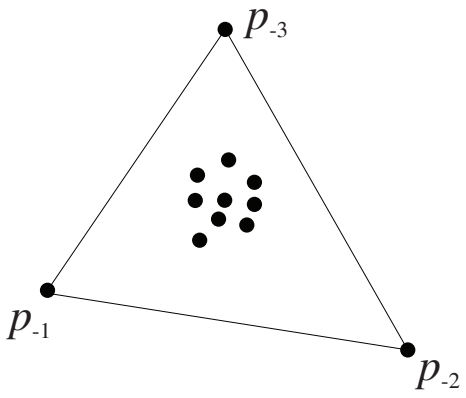
Oglejmo si pričakovano stopnjo točke v Delaunayevi triangulaciji skozi logiko algoritma. Naj bo P_r množica točk dolžine r, nad katero gradimo Delaunayevo triangulacijo s podanim algoritmom. V koraku r vstavimo v triangulacijo točko $p_r \in P_r$ in s tem najprej razdelimo en ali dva trikotnika ter tako ustvarimo tri ali štiri nove trikotnike. Ta delitev trikotnikov naredi enako število novih povezav (tri ali štiri) in za vsako povezavo, ki jo zamenjamo v proceduri *LegalizirajPovezavo*, ustvarimo še dva nova trikotnika. Po

vstavitvi p_r v triangulacijo obstaja k povezav, ki vsebujejo p_r. Stopnja točke p_r je tako enaka k. Skupaj smo v koraku r ustvarili $2k-3$ novih trikotnikov. Delaunayeva triangulacija ima lahko največ $3(r+3)-6$ povezav, kar smo izpeljali iz Eulerjeve formule v poglavju 2.2. Tri izmed teh povezav so povezave, ki tvorijo začetni umetni trikotnik in jih je potrebno odstraniti. Tako je skupna stopnja vseh točk v P_r manjša od $2[3(r+3)-9]$, kar znese $6r$. Od tod ponovno sledi, da je pričakovana stopnja naključno izbrane točke v P_r manjša od 6.

3.4 Algoritem Bowyer-Watson

Bowyer [5] in Watson [28] sta v reviji Computers Journal istočasno vsak v svojem članku predstavila naključni inkrementalni algoritem, ki kot primarni kriterij upošteva Delaunayevo pravilo, trikotnike Delaunayeve triangulacije pa tvori znoraj povezovalne metode. Izpis 4 prikazuje psevdokodo algoritma.

Oglejmo si delovanje algoritma bolj podrobno. Algoritem se prične s tvorbo dovolj velikega umetnega trikotnika, ki zajame vse točke iz podane množice P. Umetne točke p_{-1}, p_{-2} in p_{-3} postavimo dovolj daleč, da ne vplivajo na zgradbo Delaunayeve triangulacije nad P, kot to prikazuje slika 3.9. Tako gradimo Delaunayevo triangulacijo nad $P \cup \{p_{-1}, p_{-2}, p_{-3}\}$.



Slika 3.9: Veliki umetni trikotnik.

Izpis 4 Algoritem: BowyerWatson(P)

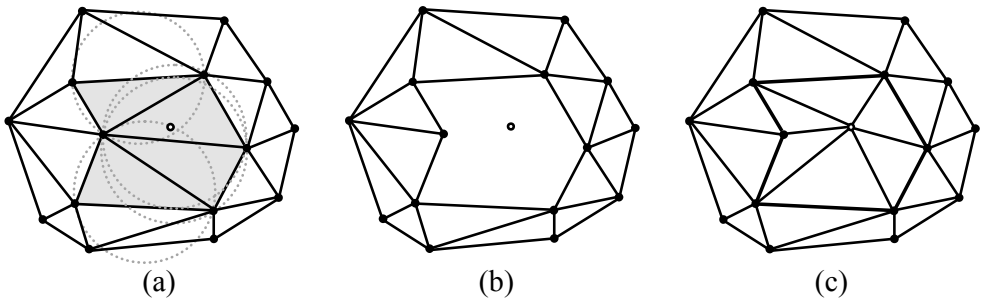
Vhod. Množica P dolžine $n + 1$ točk v ravnini.

Izhod. Delaunayeva triangulacija nad P.

```
1: Inicializiraj seznam trikotnikov.
2: Določi veliki umetni trikotnik.
3: Dodaj vozlišča umetnega trikotnika na konec P.
4: Dodaj umetni trikotnik v seznam trikotnikov.
5: za vsako točko iz P
6:     Inicializiraj seznam povezav.
7:     za vsak trikotnik v seznamu trikotnikov
8:         Izračunaj središče očrtane krožnice in njen polmer.
9:         če točka leži znotraj očrtane krožnice
10:            potem Dodaj tri povezave trikotnika v seznam povezav.
11:                  Odstrani trikotnik iz seznama trikotnikov.
12:    Izbriši vse podvojene povezave iz seznama povezav.
                                  ▷ Ostanejo samo povezave na robu luknje.
13:    Dodaj na seznam trikotnikov vse trikotnike, ki nastanejo s povezavami med točko in točkami, ki ležijo na robu luknje.
14: Odstrani iz seznama trikotnikov vse trikotnike, ki imajo skupno vozlišče z umetnim trikotnikom.
15: Odstrani vozlišča umetnega trikotnika iz P.
16: vrni seznam trikotnikov
```

Z določitvijo velikega umetnega trikotnika smo zagotovili, da se v njegovi notranjosti nahajajo vse točke iz P. Nadalje algoritem naključno izbere točko iz P in jo vstavi v triangulacijo. Za vsako vstavljeno točko najprej poiščemo pripadajoči trikotnik ali trikotnika, nato pa izmed vseh obstoječih trikotnikov v triangulaciji določimo tiste, katerih očrtana krožnica vsebuje na novo vstavljeno točko. Ti trikotniki niso več Delaunayevi, zato jih zbrišemo iz triangulacije, na njihovem mestu pa nastane luknja. Algoritem ustvari povezave med novo točko in točkami, ki ležijo na mejnih povezavah luknje.

Tako tvorimo nove trikotnike, ki so Delaunayevi, in z njimi zapolnimo luknjo v triangulaciji (slika 3.10).



Slika 3.10: Vstavitev točke (bele barve) v $\mathcal{DT}$: vsi trikotniki, ki znotraj svoje očrtane krožnice vsebujejo novo vstavljeno točko (a), so izbrisani (b). Luknja je zapolnjena z novimi trikotniki, ki so zgrajeni s povezavami med novo točko in točkami na robu luknje [21].

V začetnem primeru bo prva vstavljena točka padla znotraj umetnega trikotnika in ga razdelila na tri manjše trikotnike, naprej pa poteka delitev trikotnikov in zamenjava luknje za preostale točke iz P iterativno, dokler ne obdelamo celotne množice P.

V zadnjem koraku iz triangulacije algoritem izbriše vse trikotnike, od katerih vsaj eno oglišče tvori točka p_{-1}, p_{-2} ali p_{-3}. V tem koraku moramo biti pozorni, da ne uničimo roba konveksne ovojnice triangulacije. Rezultat je Delaunayeva triangulacija nad P.

Algoritem je razširljiv na več dimenzij, za gradnjo Delaunayeve triangulacije potrebuje $\mathcal{O}(n^{1+1/k})$ časa, kjer je k število dimenzij, in $\mathcal{O}(n)$ časa za upravljanje s podatkovno strukturo [5]. Nas zanima dvodimenzionalna Delaunayeva triangulacija, za katero je pričakovana časovna zahtevnost algoritma $\mathcal{O}(n^{1,5})$. V kolikor bi točke pred vstavitvijo v algoritem sortirali, bi algoritem za gradnjo Delaunayeve triangulacije potreboval $\mathcal{O}(n \log n)$ časa. V praksi se izkaže, da je to navadno zanemarljivo.

Poglavje 4

Implementacija algoritma

V tem poglavju bomo podrobno opisali implementacijo našega algoritma in pojasnili način merjenja izvajalnega časa algoritma. Predstavili bomo način izračuna najvišje in povprečne stopnje točke v Delaunayevi triangulaciji, ki jo zgradi naš algoritem. Pojasnili bomo način generiranja testnih vzorcev, nad katerimi bomo z našim algoritmom gradili Delaunayevo triangulacijo.

4.1 Implementacija algoritma

V programskem jeziku Java smo razvili algoritem za izgradnjo Delaunayeve triangulacije v ravnini po vzoru algoritma Bowyer-Watson.

Najprej je bilo potrebno izbrati primerno podatkovno strukturo, ki nam bo omogočala gradnjo Delaunayeve triangulacije in hrambo trikotnikov. Odločili smo se za DAG, saj nam omogoča enostavno navigacijo med vsebovanimi vozlišči. Uporabili smo eno izmed klasičnih implementacij v Javi z razredom *HashMap*, kot vozlišča pa smo shranjevali trikotnike, za katere smo ustvarili lasten razred.

Program najprej prebere in shrani točke iz podane datoteke. V ta namen smo ustvarili razred *Tocka*, kateremu pripadata dve koordinati in ustrezne metode za inicializacijo nove točke, določanje lokacije ter merjenje razdalje. Nad množico točk naredimo naključno permutacijo.

Za inicializacijo algoritma ustvarimo začetni trikotnik z umetnimi točkami. Trikotnike shranjujemo kot lasten razred *Trikotnik*, sestavljen iz treh točk, ki določajo oglišča trikotnika in metod, ki določajo, ali se posamezna točka nahaja znotraj danega trikotnika in znotraj očrtane krožnice. Metode bomo opisali podrobneje v nadaljevanju. Za umetne točke smo izbrali točke s koordinatami $(-3maxX, -3maxY)$, $(3maxX, 0)$ in $(0, 3maxY)$, pri čemer je $maxX$ absolutna vrednost prve koordinate tiste točke iz podane množice, ki ima največjo absolutno x-koordinato, $maxY$ pa je absolutna vrednost druge koordinate tiste točke, ki ima največjo absolutno y-koordinato. S tem smo zagotovili, da začetni trikotnik zaobjame vse točke iz množice.

Začetni trikotnik vstavimo v triangulacijo in s tem poženemo algoritem. Ustvarili smo lasten razred *Triangulacija*, ki hrani DAG in vsebuje večji del logike za gradnjo Delaunayeve triangulacije. Algoritem shrani začetni trikotnik v DAG. V triangulacijo nato iterativno vstavljamo točke iz množice.

V prvem koraku vstavljeni točki poiščemo trikotnik, znotraj katerega se nahaja. To naredimo z metodo *znotraj*, ki se sprehodi čez trikotnike v DAG in preverja, ali posamezen trikotnik zadošča kriteriju točke znotraj trikotnika, dokler ne najde ustreznega trikotnika. Ustvarili smo metodo, ki vozliščem v trikotniku zgradi poligon preko javanskega razreda *Polygon*, nato pa z metodo *contains* iz pripadajočega javanskega razreda preverja, ali se točka nahaja znotraj tega poligona ali ne. Metoda upošteva tudi primer, ko se točka nahaja na obstoječi povezavi trikotnika. Kot rezultat metoda *znotraj* vrne trikotnik, znotraj katerega se nahaja točka. V primeru, da se točka nahaja na povezavi, ki pripada dvema trikotnikoma, metoda vrne samo trikotnik, ki ga odkrije najprej. To zadostuje, saj v naslednjem koraku preverimo, kateri trikotniki v trenutni triangulaciji ne ustrezajo kriteriju prazne očrtane krožnice in tako obravnavamo drugi trikotnik v naslednjem koraku.

V drugem koraku poiščemo luknjo, ki jo povzroči vstavitev nove točke. To naredimo znotraj metode *poisciSivino*, ki kot argument prejme novo vstavljeno točko in trikotnik, znotraj katerega se ta točka nahaja. V metodi pregledamo DAG za vse trikotnike, ki vsebujejo novo točko v svoji očrtani

krožnici. Za preverjanje tega kriterja smo v razredu *Trikotnik* naredili metodo *znotrajOcrtaneKroznice*. Ta kot argument prejme točko, kot rezultat pa vrne odgovor ali se točka nahaja znotraj podanega trikotnika ali ne. Pogoj preverjamo tako, da primerjamo razdaljo med središčem očrtane krožnice in novo točko in če je ta manjša od polmera, točka leži znotraj očrtane krožnice trikotnika. Središče očrtane krožnice izračunamo s pomočjo kartezijskih koordinat in vektorskega produkta po enačbah:

$$U_x = [(A_x^2 + A_y^2)(B_y - C_y) + (B_x^2 + B_y^2)(C_y - A_y) + (C_x^2 + C_y^2)(A_y - B_y)]/D,$$
$$U_y = [(A_x^2 + A_y^2)(C_x - B_x) + (B_x^2 + B_y^2)(A_x - C_x) + (C_x^2 + C_y^2)(B_x - A_x)]/D,$$

pri čemer je $D = 2[A_x(B_y - C_y) + B_x(C_y - A_y) + C_x(A_y - B_y)]$.

Točka s koordinatami (U_x, U_y) predstavlja središče očrtane krožnice trikotnika. Polmer očrtane krožnice dobimo z izračunom razdalje vektorja med enim izmed krajišč trikotnika in središčem očrtane krožnice.

Vsak trikotnik, za katerega identificiramo, da točka leži znotraj njegove očrtane krožnice, shranimo. Metoda *znotrajOcrtaneKroznice* na koncu vrne seznam vseh trikotnikov, ki tvorijo luknjo.

Tretji korak je korak tvorjenja trikotnikov z legalnimi povezavami. To naredimo z metodo *legalizirajPovezave*, ki kot argument prejme novo vstavljeno točko in seznam trikotnikov, ki tvorijo luknjo po vstavitvi točke v triangulacijo. Za vsak trikotnik, vsebovan v luknji, pregledamo povezave in iz luknje izbrišemo nelegalne povezave, legalne pa na novo ustvarimo. Iz DAG odstranimo vse trikotnike v luknji, nato pa z legalnimi povezavami tvorimo nove trikotnike, ki so Delaunayevi, in jih vstavimo v DAG.

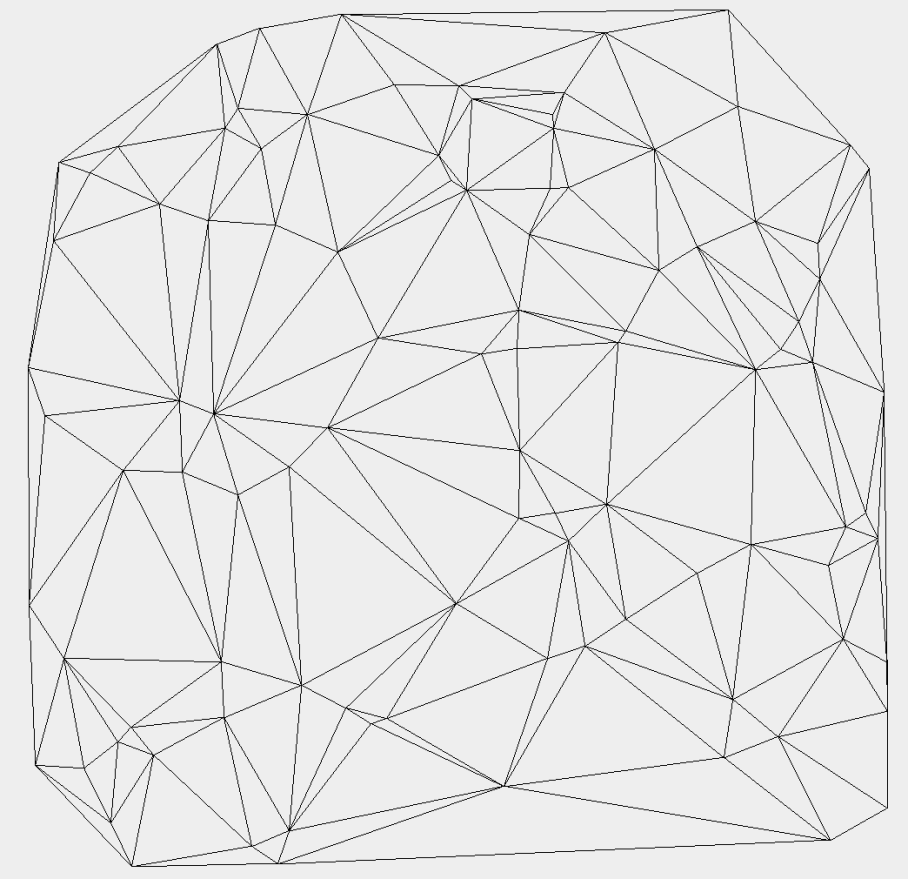
Ponavljamo prvi, drugi in tretji korak, dokler ne obdelamo celotne množice točk. Nato odstranimo trikotnike, ki vsebujejo vsaj eno izmed začetnih umetnih točk. To naredimo z metodo *odstraniDummyTocke*, ki kot argument prejme začetni trikotnik. V metodi se sprehodimo čez vse trikotnike v DAG in odstranimo trikotnike, katerih vsaj eno krajišče je enako krajišču začetnega trikotnika. S tem smo poskrbeli za odstranitev umetnih točk, posledično pa

smo v tem koraku lahko uničili rob konveksne ovojnice triangulacije, zato je potrebno preveriti, ali se vse povezave Delaunayeve triangulacije, ki ležijo na robu konveksne ovojnice, res nahajajo v DAG. V ta namen najprej z metodo *poisciRobKonveksneOvojnice* poiščemo rob konveksne ovojnice triangulacije. Metoda je implementacija algoritma *QuickHull* za iskanje roba konveksne ovojnice nad končno množico točk v ravnini. Kot argument prejme metoda seznam podanih točk, kot rezultat pa vrne seznam točk, ki tvorijo rob konveksne ovojnice. Za vsak par sosednjih točk, ki jih vrne metoda *poisciRobKonveksneOvojnice*, preverimo, ali se povezava, ki vsebuje ta par točk, nahaja v DAG. Če ugotovimo, da določena povezava manjka, identificiramo manjkajoči trikotnik, del katerega je manjkajoča povezava, in nato trikotnik dodamo v DAG. S tem korakom se algoritem zaključi in kot rezultat dobimo DAG, ki vsebuje samo trikotnike Delaunayeve triangulacije nad podanimi točkami.

Na koncu pripravimo še grafični izris Delaunayeve triangulacije. Za vsak trikotnik v DAG izrišemo povezave med krajišči trikotnika in tako dobimo končno sliko. Primer grafičnega izrisa je prikazan na slikah 4.1 in 4.2.

4.2 Merjenje izvajalnega časa

Za pričetek merjenja izvajalnega časa algoritma smo izbrali korak, kjer naredimo naključno permutacijo nad prebranimi točkami. Na tem mestu shranimo trenutni čas najbolj natančne možne sistemske ure z javansko metodo *System.nanoTime*. Metoda shrani časovno vrednost v nanosekundah. Za konec merjenja izvajalnega časa algoritma smo določili korak, ko algoritem preveri, ali so vse povezave s točkami na robu konveksne ovojnice triangulacije prisotne v DAG in če niso, tvori pripadajoče trikotnike. V tem koraku algoritem vrne Delauayevo triangulacijo in s tem zaključi svoj namen, zato smo tudi merjenje po njegovi izvršitvi ustavili. Z merjenjem zaključimo tako, da z metodo *System.nanoTime* znova shranimo trenutno vrednost, ki je od prejšnje shranjene vrednosti večja natanko za čas, ki ga je porabil algori-

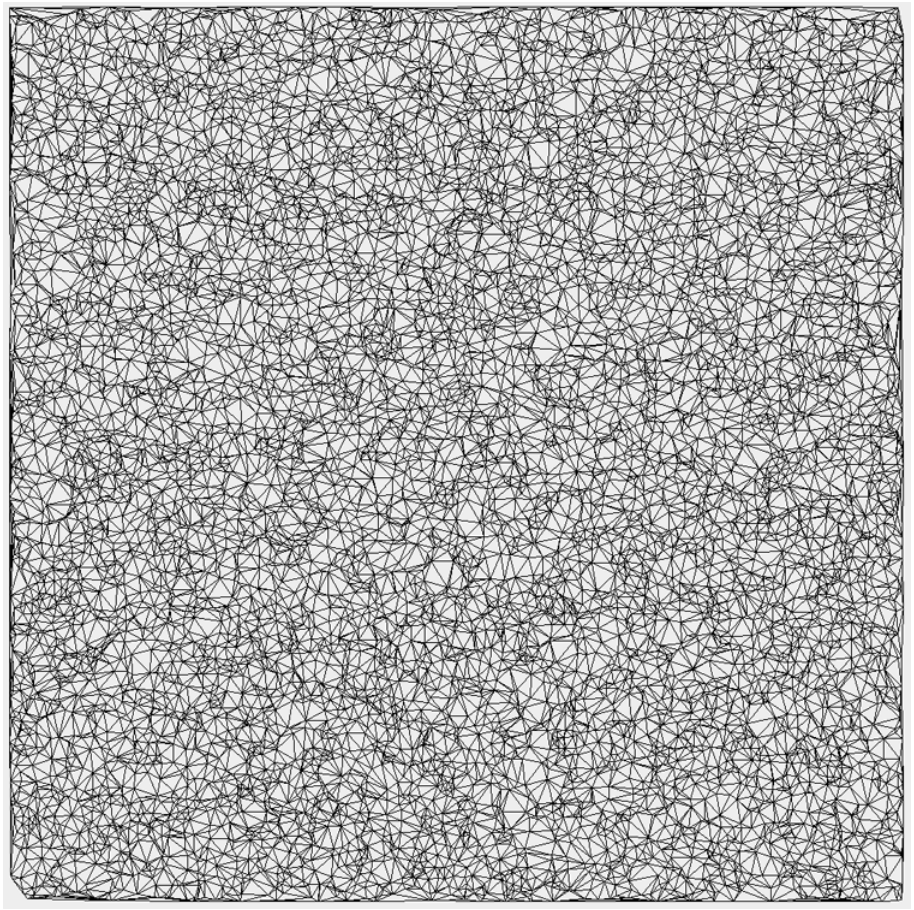


Slika 4.1: Primer grafičnega izrisa našega algoritma za množico 100 točk.

tem za gradnjo Delaunayeve triangulacije. Vrednosti med seboj odštejemo in s tem dobimo izvajalni čas algoritma v nanosekundah, ki ga zaradi boljše preglednosti pretvorimo v milisekunde in sekunde.

4.3 Določanje najvišje in povprečne stopnje točke

Najvišjo in povprečno stopnjo točke določamo po tem, ko algoritem že zgradi Delaunayevo triangulacijo nad podano množico točk. Definirali smo tabelo, ki bo hranila stopnjo točke za vsako točko iz podane množice. Iterativno



Slika 4.2: Primer grafičnega izrisa našega algoritma za množico 10000 točk.

se sprehodimo čez vse trikotnike, ki gradijo Delaunayevo triangulacijo in za vsako točko v tabelo shranjujemo njeno stopnjo. To naredimo tako, da preverjamo število pojavitev točke v trikotnikih v DAG. Vsakič, ko se točka nahaja v enem izmed Delaunayevih trikotnikov, povečamo števec njene stopnje v pripadajočem polju v tabeli, dokler ne obdelamo vseh točk in trikotnikov. Posebni primeri so točke, ki ležijo na robu konveksne ovojnice, te imajo stopnjo za ena večjo kot je število trikotnikov, ki jim pripadajo, zato smo dodali za te točke še poseben kriterij, ki preveri, ali je posamezna točka del roba konveksne ovojnice Delaunayeve triangulacije in če je, ustrezno povečamo stopnjo pripadajoči točki še za ena.

Najvišjo stopnjo točke dobimo tako, da poiščemo najvišjo vrednost v tabeli stopenj točk. Iterativno primerjamo vrednosti in vedno shranjujemo največjo, dokler ne obdelamo vseh točk. Povprečno stopnjo točke dobimo tako, da seštejemo vse stopnje točk med seboj in vsoto delimo s številom točk.

4.4 Generator naključnih vzorcev točk

Za namene testiranja našega algoritma smo v programskem jeziku Java naredili program *GeneratorTock*. Program kreira novo datoteko, v katero preko javanskega razreda *Random* zapiše izbrano število točk z naključno generirano vrednostjo koordinat na podanem intervalu. Tako vsakič dobimo naključen vzorec točk izbrane velikosti na podanem intervalu, nad katerim bomo gradili Delaunayevo triangulacijo.

Poglavje 5

Rezultati

Poglavje je namenjeno predstavitvi rezultatov. Opisali bomo testno okolje in izbiro testnih vzorcev. Ogledali si bomo čas izvajanja algoritma in določili najvišjo ter povprečno stopnjo točke v Delaunayevi triangulaciji, ki jo zgradi naš algoritem.

5.1 Testno okolje in testni vzorci

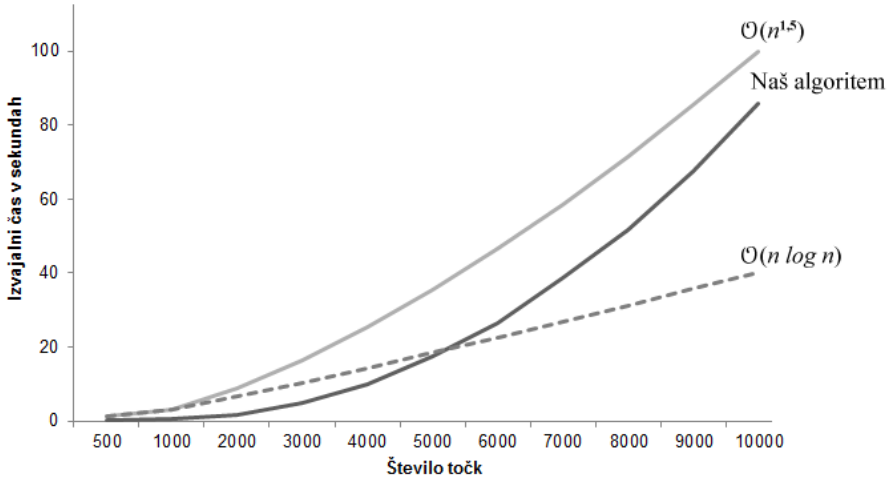
Algoritem smo poganjali na računalniku s procesorjem Intel Pentium E6600 Dual-Core 3.06 GHz s 4 GB delovnega pomnilnika, v 64-bitnem operacijskem sistemu Windows 7 Professional, kot testno okolje pa smo uporabili razvojno okolje NetBeans. Za testiranje našega algoritma smo pripravili sto naključnih vzorcev za množice 10, 50, 100, 500, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000 in 10000 točk, razpršenih na intervalu $[1, 0] \times [0, 1]$, nad katerimi smo zgradili Delaunayevo triangulacijo.

5.2 Izvajalni čas algoritma

Testirali smo izvajalni čas algoritma. V tabeli 5.1 so predstavljeni rezultati za testne vzorce, slika 5.1 pa prikazuje izvajalni čas algoritma v odvisnosti od števila točk.

Število točk	Povprečni čas izvajanja (v sekundah) pri 100 vzorcih
10	0,018
50	0,067
100	0,111
500	0,252
1000	0,512
2000	1,682
3000	4,879
4000	10,036
5000	17,580
6000	26,621
7000	38,550
8000	51,829
9000	67,616
10000	85,744

Tabela 5.1: Izvajalni čas algoritma pri podanem številu točk.



Slika 5.1: Izvajalni čas algoritma v odvisnosti od števila točk.

Opazimo, da povprečni čas izvajanja algoritma narašča s številom točk, ostaja pa znotraj meje $\mathcal{O}(n^{1,5})$.

5.3 Najvišja stopnja točke

Poiskali smo najvišjo stopnjo točke v Delaunayevi triangulaciji nad podanim vzorcem. Tabela 5.2 prikazuje rezultate za povprečno vrednost najvišje stopnje, v tabeli 5.3 pa so predstavljene maksimalne in minimalne najvišje stopnje točk izmed vseh vzorcev pri podanem številu točk.

Število točk	Povprečna vrednost najvišje stopnje točke pri 100 vzorcih	Pričakovana vrednost najvišje stopnje točke ($\log^2 n$)
10	6,56	1
50	8,97	2,89
100	9,54	4
500	11,05	7,28
1000	11,49	9
2000	12,12	10,90
3000	12,65	12,09
4000	12,90	12,97
5000	13,08	13,68
6000	13,37	14,27
7000	13,69	14,78
8000	14,22	15,23
9000	14,34	15,64
10000	14,51	16

Tabela 5.2: Najvišja stopnja točk pri podanem številu točk.

Z večanjem števila točk opazimo postopno višanje vrednosti najvišje stopnje točke. Opazimo, da vzorci do vključno z 2000 točkami krepko presegajo teoretično pričakovano najvišjo stopnjo, vzorci od 4000 točk naprej pa v

Število točk	Maksimalna vrednost najvišje stopnje točke pri 100 vzorcih	Minimalna vrednost najvišje stopnje točke pri 100 vzorcih
10	9	6
50	12	8
100	12	8
500	14	9
1000	14	10
2000	14	11
3000	16	11
4000	17	11
5000	18	11
6000	18	12
7000	19	12
8000	19	12
9000	20	12
10000	21	12

Tabela 5.3: Maksimalna in minimalna vrednost najvišje stopnje točk pri podanem številu točk.

povprečju padejo pod pričakovano mejo. Prav tako opazimo, da je med 100 vzorci za vsako število točk obstajal takšen vzorec, kjer je najvišja stopnja presegla pričakovano mejo.

5.4 Povprečna stopnja točke

Testirali smo povprečno stopnjo točke v Delaunayevi triangulaciji nad podanim vzorcem. Rezultati so predstavljeni v tabeli 5.3.

Z večanjem števila točk opazimo rast povprečne stopnje točke, ki se asimptotsko bliža meji 6.

Število točk	Povprečna vrednost povprečne stopnje točke pri 100 vzorcih
10	5,0460
50	5,7026
100	5,8211
500	5,9452
1000	5,9653
2000	5,9782
3000	5,9821
4000	5,9833
5000	5,9851
6000	5,9853
7000	5,9859
8000	5,9863
9000	5,9867
10000	5,9872

Tabela 5.4: Povprečna stopnja točk pri podanem številu točk.

Poglavje 6

Razprava in sklepne ugotovitve

Predstavljena diplomska naloga obravnava področje Delaunayevih triangulacij v ravnini. Osredotočili smo se na gradnjo Delaunayeve triangulacije z inkrementalnim konstrukcijskim algoritmom. Cilji diplomske naloge so bili lastna implementacija algoritma, meritev izvajalnega časa in preverjanje najvišje ter povprečne stopnje točk v Delaunayevi triangulaciji, ki jo je zgradil naš algoritem. Zadane cilje smo izpolnili.

Časovne meritve izvajanja našega algoritma so pokazale, da se dejanski izvajalni čas nahaja znotraj teoretičnega. Vseeno pa izbira DAG za podatkovno strukturo ni najbolj optimalna izbira za inkrementalne konstrukcijske algoritme. Guibas in Stolfi [18] sta predstavila podatkovno strukturo *quad-edge*, ki je namenjena triangulaciji točk v dvo- in trodimenzionalnih prostorih. O. Devillers [11] je predstavil podatkovno strukturo *Delaunayeva hierarhija*, ki deluje na principu lokacije točke glede na najbližnje sosede. Struktura v primerjavi z DAG zavzema dosti manjši odstotek pomnilnika. Nadgradnja našega algoritma z izboljšano podatkovno strukturo bi optimizirala porabo pomnilnika, bi se pa s tem povečala kompleksnost algoritma.

Broutin, Devillers in Hemsley so izpeljali teoretično mejo za asimptotično obnašanje maksimalne stopnje točke v Delaunayevi triangulaciji, ko število točk narašča [6]. Analiza rezultatov naših meritev je pokazala, da bo v praksi najvišja stopnja točke v povprečju ostala pod pričakovano mejo pri velikem

številu točk, ki ga pri izbranem vzorcu ocenjujemo na 4000 točk.

Teoretična pričakovana vrednost povprečne stopnje točke naj ne bi presegla vrednosti 6. Rezultati naših meritev to potrjujejo. Z večanjem števila točk se v praksi vrednost povprečne stopnje točke asimptotsko bliža pričakovani vrednosti 6 in je ne preseže.

Ugotovili smo, da se z večanjem števila točk postopoma višata tako najvišja kot povprečna stopnja točke. Ker so vzorci točk razporejeni na istem intervalu, iz tega lahko sklepamo, da sta najvišja in povprečna stopnja točke odvisni od gostote razporeditve točk. Več kot je točk razpršenih na istem intervalu, višja sta tudi najvišja in povprečna stopnja točke.

V praksi je včasih potrebno graditi triangulacije nad množico točk, ki ne omogoča gradnje kvalitetnih trikotnikov z maksimizacijo ozkih kotov, zato vključimo v postopek triangulacije dodatne točke, imenovane *Steinerjeve točke*. Postopek nas vpelje v koncept *omejene Delaunayeve triangulacije*, ki je danes še vedno zelo aktualno področje raziskav v računski geometriji [14, 24] in lahko predstavlja predmet našega nadaljnjega preučevanja.

Literatura

[1] F. Aurenhammer in R. Klein, "Voronoi diagrams", Handbook of Computational Geometry, v: J. Sack in G. Urrutia (Ur.), Elsevier, 2000.

[2] M. Bern in D. Eppstein, "Mesh Generation and Optimal Triangulation", *Computing in Euclidean Geometry, World Scientific, v: F.K. Hwang in D.-Z. Du*, strani 201–290, 1992.

[3] M. W. Bern, D. Eppstein in F.F. Yao, "The Expected Extremes in a Delaunay Triangulation", *ICALP*, strani 674–685, 1991.

[4] M. Bostock, "U.S. Airports Voronoi", Dosegljivo: `https://bl.ocks.org/mbostock/4360892`, 2016. [Dostopano: 2016].

[5] A. Bowyer, "Computing Dirichlet Tessellations", *The Computer Journal*, 24(2):162–166, 1981.

[6] N. Broutin, O. Devillers in R. Hemsley, "The Maximum Degree of a Random Delaunay Triangulation in a Smooth Convex", *AofA 2014 - 25th International Conference on Probabilistic, Combinatorial and Asymptotic Methods for the Analysis of Algorithms*, 2014.

[7] CGAL, "Poisson Surface Reconstruction", Dosegljivo: `http://doc.cgal.org/latest/Poisson_surface_reconstruction_3/index.html`, 1995-2016. [Dostopano: 2016].

[8] M. de Berg, O. Cheong, M. van Kreveld in M. Overmars, "Computational Geometry - Algorithms and Applications", Springer-Verlag, Berlin, 1997.

[9] B.N. Delaunay, "Sur la spèhre vide", *Bulletin de l'Académie des Sciences de l'URSS, Classe des sciences mathématiques et naturelles*, 1934(6):794–800, 1934.

[10] R. Descartes, "Principia philosophiae", *Ludovicus Elzevirius*, 1644.

[11] O. Devillers, "The Delaunay Hierarchy", *International Journal of Foundations of Computer Science*, 2002(13):163–180, 2002.

[12] R. DeWall, "A Fast Divide and Conquer Delaunay Triangulation Algorithm in Ed", *Computer-Aided Design*, 30(5):333—-341, 1998.

[13] D. Ding-Zhu in F. Hwang, "Computing in Euclidean Geometry", Scientific Publishing Co. Pte. Ltd, Singapur, 1992.

[14] V. Domiter, "Constrained Delaunay Triangulation Using Plane Subdivision", *Proceedings of the 8th central European seminar on computer graphics*, strani 105–110, 2004.

[15] L. Euler, "Elementa doctrinae solidorum – demonstratio nonnullarum insignium proprietatum, quibus solida hedris planis inclusa sunt praedita", *Novi comment acad. sc. imp. Petropol.*, 4(3):109–140–160, 1752.

[16] S. Fortune, "A Sweep-line Algorithm for Voronoi Diagrams", *Algorithmica*, 2(2):153–174, 1987.

[17] L. Guibas, D. Knuth in M. Sharir, "Randomised Incremental Construction of Delaunay and Voronoi Diagrams", *Algorithmica*, 7(4):381–413, 1992.

[18] L. Guibas in J. Stolfi, "Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams", *ACM Transactions on Graphics*, 2(4):74–123, 1985.

[19] C.L. Lawson, "Triangulation of Plane Point Sets", *Jet Propulsion Laboratory Space Programs Summary 37-35*, 4:24–25, 1965.

[20] C.L. Lawson, "Transforming Triangulations", *Discrete Mathematics*, 1972(3):365–372, 1972.

[21] H. Ledoux, "Modelling Three-dimensional Fields in Geoscience with the Voronoi Diagram and its Dual", Doktorska dizertacija, School of Computing, University of Glamorgan, 2006.

[22] H. Ledoux, "Computing the 3d Voronoi Diagram Robustly: An Easy Explanation", *In Proceedings of the 4th International Symposium on Voronoi Diagrams in Science and Engineering*, str:117–129, 2007.

[23] D.T. Lee in B.J. Schachter, "Two Algorithms for Constructing a Delaunay Triangulation", *International Journal of Geographic Information Science*, 9(3):219–242, 1980.

[24] M. Qi, T.T. Cao, in T.S. Tan, "Computing 2D Constrained Delaunay Triangulation using the GPU", *IEEE Transactions on Visualization and Computer Graphics*, 19(5):736–748, 2012.

[25] C. Siu-Wing, T.K. Dey in J Shewchuk, "Delaunay Mesh Generation", Taylor and Francis Group, LLC, Združene Države Amerike, 2013.

[26] M. Teillaud, "Towards Dynamic Randomized Algorithms in Computational Geometry", Springer-Verlag, Berlin, 1993.

[27] G.M. Voronoi, "Nouvelles applications des paramètres continus à la théorie des formes quadratiques", *Journal für die reine und angewandte Mathematik*, 1908(134):198–287, 1908.

[28] D.F. Watson, "Computing the N-Dimensional Delaunay Tessellation with Application to Voronoi Polytopes", *The Computer Journal*, 24(2):167–181, 1981.

[29] B. Žalik, "An Efficient Sweep-line Delaunay Triangulation Algorithm", *Computer-Aided Design*, 37(10):1027–1038, 2005.